HARDWARE DYNAMICAL SYSTEM FOR SOLVING OPTIMIZATION PROBLEMS

A Dissertation
Presented to
The Academic Faculty

By

Muya Chang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
Georgia Institute of Technology
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2020

HARDWARE DYNAMICAL SYSTEM FOR SOLVING OPTIMIZATION PROBLEMS

Thesis committee:

Dr. Arijit Raychowdhury, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Muhannad S Bakir
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Justin Romberg
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Shreyas Sen
School of Electrical and Computer Engineering
Purdue University

Dr. Tushar Krishna
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Keith Bowman
Principal Engineer
Qualcomm

Date approved: November 20, 2020

God, grant me the serenity to accept the things I cannot change,

courage to change the things I can,

and wisdom to know the difference.

Reinhold Niebuhr, "Serenity Prayer"

Thank you to my academic adviser who guided me in this process, my family who have always supported me, and Xueyang who has been so important that I wouldn't be able to finish this without all of them.

ACKNOWLEDGMENTS

First of all, this work would not have been possible without the financial support of the Semiconductor Research Corporation, National Science Foundation, and Nanoelectronics Research Corporation.

I am very grateful to all of those with whom I have had the pleasure to work during this and other related projects. Each of the members of my Dissertation Committee has provided me extensive personal and professional guidance and taught me a great deal about both scientific research and life in general.

I would especially like to thank Dr. Arijit Raychowdhury. As my advisor and mentor, he has taught me more than I could ever give him credit for. He has shown me, by his example, what a good researcher (and person) should be. His mentorship was pivotal in expanding my perspectives on research and insights in personal and professional development.

I would like to thank many of my lab mates, such as Samantak who guided me during my early PhD life, Ningyuan whom I met first upon arriving and has been like a brother to me ever since, Brian who is brilliant and received the Qualcomm Fellowship with me together, Jonghyeok who is a great researcher and we have had a lot of nice coffee time together.

I would also like to thank my friends, especially Li-Hsiang Lin, Ruei-Bao Wu, Sheng-Chuan Hwang, Nealson Li, Kuan-Yu Li, Yen Kuo, Mike Hsueh, Yi-Han Lu, for the fun and memorable time together. I am extremely fortunate to have such wonderful people around me during my Ph.D. years.

Last but not the least, nobody has been more important to me in the pursuit of this journey than the members of my family. I would like to thank my parents, whose love and guidance are with me in whatever I pursue. They are the ultimate role models.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

ADMM alternating direction method of multipliers

ALU arithmetic logic unit

ASIC application-specific integrated circuit

BEOL back-end-of-line

CAD computer-aided design

CMI compressible microinterconnect

CMOS complementary metal–oxide–semiconductor

CNF conjunctive normal form

CNN convolutional neural networks

CPU computing unit

CRAM compute SRAM

CT computerized tomography

CTDS continuous-time dynamical system

DCO digital controlled oscillator

DFT Discrete Fourier transform

DR dynamic range

DRAM dynamic random-access memory

DSP digital signal processing

FFT Fast Fourier transform

FPGA field-programmable gate array

GALS globally asynchronous and locally synchronous

HDD hard disk drive

IC integrated circuits

LOT lapped orthogonal transform

MAC multiply-accumulation

MRI magnetic resonance imaging

OPU optimization processing units

PIM processing in memory

PSNR peak signal-to-noise ratio

PVT process, voltage, and temperature

RS row stationary

SAT boolean satisfiability

SGD stochastic gradient descent

SPICE simulation program with integrated circuit emphasis

SRAM static random-access memory

SSD solid-state drive

SVM support vector machine

SUMMARY

Optimization problems form the basis of a wide gamut of computationally challenging tasks in signal processing, machine learning, resource planning and so on. Out of these, convex optimization, and in particular least square optimization, covers a vast majority; and recent advances in iterative algorithms to solve such problems of large dimensions have gained traction. Multi-core designs with systolic or semi-systolic architectures can be a key enabler for implementing discrete dynamical systems and realize massively scalable architectures to solve such optimization algorithms.

In the first part of the thesis, we propose a platform architecture implemented in programmable field-programmable gate array (FPGA) hardware to solve a template problem in distributed optimization, namely signal reconstruction from non-uniform sampling. This is a quintessential problem with wide-spread applications in signal processing, computational imaging etc. We expect such an architectural exploration to open up promising opportunities to solve distributed optimizations that are becoming increasingly important in real-world applications. The complete system design, mapping and optimization into an FPGA architecture as well as analysis of convergence and scalability have been presented.

In the second part of the thesis, we present OPTIMO, a 65nm, 16-b, fully-programmable, spatial-array processor with 49-cores and a hierarchical multi-cast network for solving distributed optimizations via the alternating direction method of multipliers (ADMM). ADMM is a projection based method for solving generic constrained optimizations problems. In essence, it relies upon decomposing the decision vector into subvectors, updating sequentially by minimizing an augmented Lagrangian function, and eventually updating the Lagrange multiplier. The ADMM algorithm has typically been used for solving problems in which the decision variable is decomposed into *two or multiple subvectors*. We demonstrate six template algorithms and their applications and we measure a peak energy-efficiency of 279 GOPS/W.

In the last part, instead of in discrete we focus on the optimization problems in continuous time domain. We present AC-SAT, an analog based circuits using traditional complementary metal–oxide–semiconductor (CMOS) technology for solving a representative NP-complete optimization problem, the boolean satisfiability (SAT) problem. AC-SAT is based on the deterministic continuous-time dynamical system (CTDS) and finds SAT solutions in analog polynomial time with the expense of auxiliary variables growing exponentially when needed. The overall design is programmable, modular, and has been validated through multiple stages, from high level simulation on the general purpose computing unit (CPU), low level simulation through simulation program with integrated circuit emphasis (SPICE), all the way to the measurement on the fabricated chip. Through the measurement result, we demonstrate the relationship between optimization hardness as transient chaos and show that this architecture is highly scalable and configurable.

CHAPTER 1
LITERATURE SURVEY

1.1 Optimization Problems

In the era of big data and machine learning over large data sets, the role of solving optimization problems is becoming ever important [1, 2, 3]. This is coupled with the realization that the conventional Von-Neumann machine is a poor choice for solving iterative algorithms, where large amount of data need to be periodically read and written from an external memory. In structure, a vast majority of optimization problems are iterative and involve (1) local updates of state variables with (2) near neighbor interactions to pass information until convergence is achieved [4]. Such a computing paradigm is inspired by nature including the human brain, where local computation is coupled with near-neighbor communication to solve computationally 'hard' problems [5]. An example of such algorithm is ADMM, where [4] provides a very cohesive and detailed discussion on the algorithm along with many examples in high level language such as Matlab.

1.2 Architecture Candidates

Precedence for such a computing paradigm can be found in many perspectives, such as systolic arrays of parallel computing units where each processing element is connected only to its adjacent neighbors [6, 7, 8, 9, 10], spatial arrays processing where the computing elements are spatially separated and has been widely used in signal processing as well as in machine learning [11, 12, 13], near-memory-computing where the memory is tightly coupled with the computing element to alleviate high memory access latency, or further more putting the computing inside the memory, which has also gained a lot attention lately [14, 15, 16, 17]. The research topics I'm explicitly interested in is the conjunction of hardware

dynamics and optimization [18, 19], from implementing on FPGA to designing digital or mixed-signal application-specific integrated circuit (ASIC), to solve different kinds of optimization problems, from convex optimization problems [4] to NP-complete problems [20, 21].

1.3 Systolic Array

Systolic designs take advantage of globally asynchronous and locally synchronous (GALS) architectures to reduce clocking complexity and improve design scalability [10]. This architecture finds applications in wide range of problems due to modularity, design simplicity and reduced communication cost [22, 23]. In recent work, FPGA-based implementations of systolic array have been used [24, 25, 26]. In signal processing field, [24] focus on computing FFT efficiently with high signal-to-noise ratio, and supports non-power-of-two Discrete Fourier transform (DFT). Comparing to the other pipelined and memory-based Fast Fourier transform (FFT), the authors see at least 37% of improvement on throughput. [25] aimed to achieve high-throughput QR decomposition and utilized a new 2-D systolic array architecture based on a CORDIC algorithm. Comparing to other previous proposals for FPGA, they claim to achieve at least 50% more throughput with less resource utilization. [8] proposed a semi-systolic semi-scanned array architecture for four dimensional IIR filters with Xilinex Virtex-6 FPGA. A lot of research has also been done in Machine Learning field, [9] took the fact that while FPGA are commercially available for implementing convolutional neural networks (CNN) accelerators, the current FPGA computer-aided design (CAD) tools are unable to synthesize and layout the arrays properly, therefore the authors aimed to solve the frequency degradation problems. As the result, the authors claim to be able to achieve 1.29x higher frequency with 1.5TOPS for the VGG16 network on the Xilinx KCU1500 platform. Algorithm wise the history of research is fairly longer, [26] described a systolic array for performing recursive least-squares minimization, which performs an orthogonal triangularization of the data matrix using a pipelined sequence of given rotations.

Systolic design is also commonly used in many other fields such as cryptosystems or finite field inversion [7, 6].

1.4 Spatial Array

Spatial array is in some sense relates to systolic array, however in many cases spatial array doesn't necessarily have to be in 2-D fashion, such as radar arrays, sonar arrays ... etc. On the other hand systolic architecture is in 2-D for most of the time and is believed to originate from the fact that blood circulates to and from the heart. [11] is one of the most representative research on Deep CNN using 168 spatial array processors, where the authors optimize for the energy efficiency of the entire system, including the accelerator chip and off-chip DRAM for various CNN shapes. The authors also proposed a processing dataflow, called row stationary (RS), which re-configures the computation mapping of a given shape and optimizes energy efficiency by maximally reusing data locally to reduce expensive data movement. Further more, [12] is the next generation of [11], where the authors introduce a highly flexible on-chip network, called hierarchical mesh, so as to adapt to the different amounts of data reuse and bandwidth requirements of different data types, as well as the ability to process sparse data directly in the compressed domain for both weights and activations, which improves both processing speed and energy efficiency. In addition to Machine Learning field, spatial array architecture has also been widely used in communication fields due to the evolution of 5G network. [13] proposed an agile spectral-spatial front-end filtering for instinctual blocker suppression and "power-equalizing" of desired signals, for the purpose of aiding digital arrays and reducing RX/ADC dynamic range (DR).

1.5 Near/In Memory Computing

Comparing to conventional Von-Neumann architecture on most of the modern computers, where the CPU and memory/storage (dynamic random-access memory (DRAM), solid-state drive (SSD), hard disk drive (HDD)) are often separated and therefore is more ideal

for applications which require complex computation, for many new applications in Machine Learning or Deep learning which, on the other hand, are memory bound and the computation are mostly multiply-accumulation (MAC). As a result, much research on near memory architecture has shown that such architecture can alleviate the latency as well as bandwidth bottleneck without modifying too much of hardware or software configuration changes, however due to the nature of vector dot product, a lot of researchers have tried to push it even further by bringing processing to where the data locates at, also known as processing in memory (PIM). [14] claims that unlike previous works where the output bits are usually low or even just 1 bit, they proposes a 7b inputs/outputs using voltage averaging techniques, which is sufficient to maintain good accuracy for most of the popular CNN. [15] focuses on the part where process, voltage, and temperature (PVT) variation often affects the on-die training, therefore proposed the method of using stochastic gradient descent (SGD) training to compensate for PVT and data statistics variation to design a robust in-memory support vector machine (SVM) classifier. [16] proposes a compact-rule compatible twin-8T cell to reduce the area overhead, an even-odd dual channel array to double the bandwidth, and two's compliment mapping and processing unit to support both positive and negative weights. The authors also claims the work achieves the fastest CNN multi-bit MAC operations. [17] argues that a lot of computing in memory techniques are inherently restricted to a very specific application domain, while software algorithms continue to evolve rapidly, therefore they propose a general purpose hybrid in-/near-memory compute SRAM (CRAM), which combines the efficiency of in-memory computation with the flexibility and programmability necessary for evolving software algorithms. The system achieves 475MHz operation and, with all CRAMs active the system produces 30GOPS or 1.4GFLOPS on 32b operands.

1.6 Hardware for Optimization Problems

In addition to the research on hardware for Machine Learning, there are also some research on hardware for more classical optimization problems, such as CMOS annealing or SAT problem. [18] proposed a CMOS-type lsing computer, which after mapping problems onto such computers, the model will express the behavior of magnetic spins and solve the problems by ground-state search operations. Several years later, the same group proposed a improved version [19] mainly to expand the bit widths of the coefficients and increase the number of spins handled by the processor through a CMOS-AP based on the processing-in-memory approach. The authors claim that after installing in a 2x30k spin system, the CMOS-AP demonstrates the capability for multi-chip operation with energy efficiency $1.75x10^5$ higher than running scalable annealing on a CPU. Besides the CMOS annealing, hardware research on a famous NP-complete problem, SAT problem, has also been conducted. On one hand, [20] proposed a analog circuit SAT solver, which implements CTDS and is programmable for handling different problem specifications. The authors claim that it is especially effective for solving hard k-SAT problem instances. On the other hand, [21] proposed an event-based architecture which combines a parallel analogue/digital hardware and claims that such architecture can yield state-of-the-art performance on random SAT problems under reasonable assumptions.

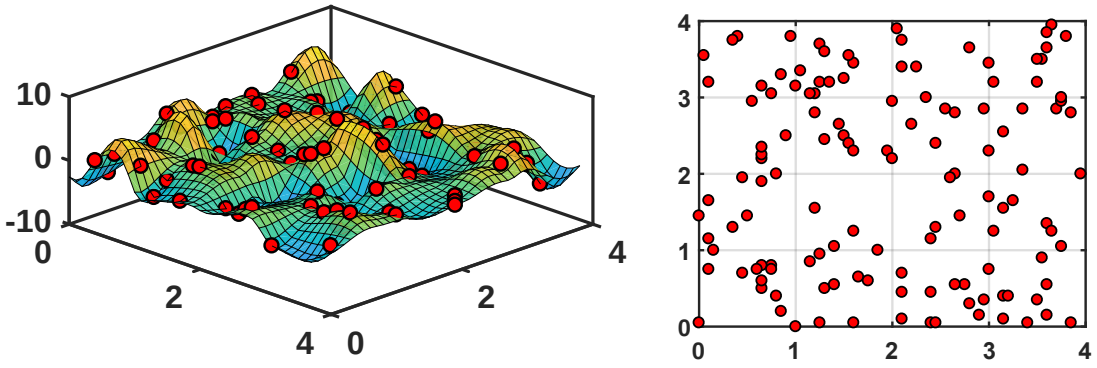
CHAPTER 2

INTRODUCTION AND BACKGROUND

2.1 Continuous Optimization: Non-Uniform Sampling and Signal Reconstruction

There are numerous applications in discrete signal processing where the process of sampling is non-uniform. It occurs when the samples cannot be collected uniformly or if samples lie in a non-uniform space [27][28]. Following the non-uniform sampling process, it is required to reconstruct the original signal. computerized tomography (CT) and magnetic resonance imaging (MRI) [29] are two such examples where reconstruction from non-uniform sampling is a fundamental step. In the following subsections, the algorithmic structure of an iterative approach to solve reconstruction from non-uniform samples is discussed.

To describe the algorithm, the following notations are used: (1) u and v are the horizontal and the vertical arguments of a continuous signal. (2) x and y are the discrete coordinate indexes. (3) ω_x and ω_y are horizontal and vertical spatial frequencies. Let $f(u, v)$ be a band-limited signal with finite energy in two dimensional real space, $\mathbb{R}^2$. The signal



(a) (b)

Figure 2.1: (a) 2D continuous function *f(u,v)* with non-uniform samples. (b) Spatial location of the non-uniform samples.

is non-uniformly sampled and are stored in vector $\boldsymbol{b}$, which are referred to as $f(x, y)$. The objective is to use the non-uniform samples to obtain complete reconstruction of $f(u, v)$. Figure 2.1 shows an example of $f(u, v)$ and the results of non-uniform sampling.

2.1.1 Signal Model

To reconstruct the signal accurately, the choice of basis functions is critical. In this work, we use 2D lapped orthogonal transform (LOT) cosine-IV harmonics as the basis functions. For each frame, a set of shifted cosine-IV basis is associated. This is how the LOT cosine-IV basis is formed. Further more, a smoothing function $g(u, v)$ is applied to all the basis functions not only to avoid distortions, but also to limit the effect of the basis on distant neighbors. (Equation 2.1) shows a general LOT cosine-IV basis function. Here, $f(u, v)$ is split into K_x by K_y frames and $[k_x, k_y]$ represent a specific frame.

$$\psi_{k_x,\omega_x,k_y,\omega_y}(u, v) = \sqrt{2} \cdot g(u - k_x, v - k_y) \cdot cos((\omega_x + \frac{1}{2})\pi(u - k_x))cos((\omega_y + \frac{1}{2})\pi(v - k_y)) \quad (2.1)$$

Since $f(u, v)$ lies in a $N_x \cdot N_y$ dimensional subspace, it can be expressed as linear combination of the basis functions as (Equation 2.2), where α refers to the coefficients associated with the basis functions.

$$f(u, v) = \sum_{\omega_x=1}^{N_x} \sum_{\omega_y=1}^{N_y} \sum_{k_x=1}^{K_x} \sum_{k_y=1}^{K_y} \alpha(k_x, \omega_x, k_y, \omega_y)\psi_{k_x,\omega_x,k_y,\omega_y}(u, v) \quad (2.2)$$

2.1.2 Gram Matrix

According to (Equation 2.2), we can write an equation for each sample and the coefficients can be found by solving the inverse-linear problem of

$$\boldsymbol{Az} = \boldsymbol{b} \quad (2.3)$$

Here $\boldsymbol{b}_{\{m,1\}}$ is the sample vector, $\boldsymbol{z}_{\{KN,1\}}$ is the coefficient vector obtained by stacking the coefficients $\alpha(k_x, \omega_x, k_y, \omega_y)$, and $\boldsymbol{A}_{\{m,KN\}}$ is referred to as the Grammian matrix.

2.1.3 Approach

There are many ways to solve $\boldsymbol{z}$, one of the well known and the straight forward approaches would be using the pseudo-inverse of A. However, this method incurs extreme penalties when the size of $\boldsymbol{A}_{\{m,KN\}}$ matrix is large. Therefore, alternatively we follow an iterative approach, the Jacobi method. A general update of $\boldsymbol{z}$ in j^{th} component at the k_{th} iteration is given as (Equation 2.4), where $\boldsymbol{B} = \boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{c} = \boldsymbol{A}^T\boldsymbol{b}$.

$$\boldsymbol{z}_j^k = \boldsymbol{B}_{jj}^{-1}(\boldsymbol{c}_j - \sum_{i \neq j} \boldsymbol{B}_{ji}\boldsymbol{z}_i^{k-1}) \tag{2.4}$$

Here the inverse of matrix B is required. Since it is used locally in the computation, we pre-compute and load the values to the block ram as part of the initial values. Per-core B matrices are small and the computation of the inverse is not computationally challenging.

Some observations are worth emphasizing: (1) To update $\boldsymbol{z}_j^k$, only the values, which are $\boldsymbol{z}_j^{k-1}$, from the previous iteration are need. (2) Columns of $\boldsymbol{A}$ are coupled only with neighboring frames, which leads to simpler computation of $\boldsymbol{B}_{ji}$.

2.2 Continuous Optimization: Alternating Direction Method of Multipliers (ADMM)

In this section, we provide an overview of ADMM as well as its distributed representation, namely *distributed ADMM*.

2.2.1 Overview

The ADMM algorithm [30] is a projection based method for solving generic problems of constrained optimizations. In essence, it relies upon decomposing the decision vector into subvectors, updating each subvector sequentially by minimizing an augmented Lagrangian

function, and finally updating the Lagrange multiplier corresponding to the constraint that couples the subvectors using a dual subgradient method. The ADMM algorithm has typically been used for solving problems in which the decision variable is decomposed into *two or multiple subvectors*. For simplicity, we only review the form of ADMM with 2 subvectors, and its generalization to the case of multiple subvectors is straightforward and is omitted here.

The original form of ADMM with 2 subvectors denoted as $x \in \mathbf{R}^n$ and $z \in \mathbf{R}^m$ solves the problem expressed as

$$\begin{aligned} &\min\ l(x) + r(z) \\ &\text{subject to } Ax + Bz = c \end{aligned} \tag{2.5}$$

where $A \in \mathbf{R}^{p \times n}$, $B \in \mathbf{R}^{p \times m}$, and $c \in \mathbf{R}^p$. We assume both $l(x)$ and $r(z)$ are convex.

We solve (Equation 2.5) using ADMM by first deriving the augmented Lagrangian function of (Equation 2.5), and it is given by:

$$L_\rho(x, z, y) = l(x) + r(z) + y^T(Ax + Bz - c) + (\rho/2)||Ax + Bz - c||_2^2 \tag{2.6}$$

where y is the Lagrange multiplier corresponding to the constraint $Ax + Bz = c$ and ρ is a positive scalar. Then, we perform an iterative algorithm which starts from arbitrary initial values $x^{(0)}$, $z^{(0)}$, and $y^{(0)}$, and update using the following updated rules:

$$\begin{aligned} x^{(k+1)} &:= \underset{x}{argmin}\ L_\rho(x, z^{(k)}, y^{(k)}) \\ z^{(k+1)} &:= \underset{z}{argmin}\ L_\rho(x^{(k+1)}, z, y^{(k)}) \\ y^{(k+1)} &:= y^k + \rho(Ax^{(k+1)} + Bz^{(k+1)} - c) \end{aligned} \tag{2.7}$$

(Equation 2.7) is solved iteratively for $k \geq 0$ until convergence is achieved.

2.2.2 Distributed ADMM

By splitting up a objective function carefully, one can transform ADMM to solve a range of useful optimization programs in a distributed fashion, and this gives rise to distributed ADMM. In its distributed form, one can parallelly solve a large optimization problem over a large data-set or a large vector over multiple cores with intermittent communication between the cores to achieve *consensus*. This makes solving many problems in image processing, signal recovery, machine learning, model prediction, and classification efficient and real-time. To provide an overview of distributed ADMM, we consider the following problem:

$$\min_{x} \; l(Ax - b) + r(x), \tag{2.8}$$

or its ADMM form:

$$\begin{aligned} &\min_{x} \; l(x) + r(z) \\ &\text{subject to } x = Az - b, \end{aligned} \tag{2.9}$$

which is a transformed representation of the original ADMM problem (Equation 2.5). There are two ways to solve (Equation 2.8) and (Equation 2.9) in a distributed manner: one is splitting across the data (or, training examples in case of model fitting), and the other is by splitting across feature vectors.

2.2.3 Template Problems

We have chosen six popular algorithms from signal/image processing and machine learning community namely as Least Square optimization [31], Lasso [32], Group Lasso [33], Elastic Net [34], Support Vector Machines [35], and Distributed Averaging [36]. The table of loss functions and the regularization functions for each template problem are shown in Table 2.1.

One thing to keep in mind is that even though all the six algorithms follow the same pro-

Table 2.1: Table for loss functions and regularization functions.

Algorithms	$f(x)$	$g(z)$	**Applications**
Least Square	$1/2\|\|Ax-b\|\|_2^2$	-	Modeling for Prediction
Lasso	$1/2\|\|Ax-b\|\|_2^2$	$\lambda\|\|x\|\|_1$	Variable selections Modeling for prediction
Elastic Net	$1/2\|\|Ax-b\|\|_2^2$	$\lambda_1\|\|x\|\|_1+\lambda_2\|\|x\|\|_2^2$	Variable selections Robust modeling for prediction
Group Lasso	$1/2\|\|Ax-b\|\|_2^2$	$\lambda\sum_{i=1}^{N}\|\|x_i\|\|_2$	Structure variable selection Modeling for prediction
Linear SVM	$\|\|x_i\|\|^2$	$y_k(a_k^T x_i+b)$	Classification
Distributed Averaging	$1/2\|\|Ax-b\|\|_2^2$	-	Large scale modeling and prediction

gram flow as Equation 2.7, how x and z are updated depends on the loss function and the regularization functions. This calls for hardware level programmability which we describe next. Further, the programming model ensures that a larger class of algorithms can be mapped to the hardware, and although we do not describe them in this research scope, the hardware architecture and the programming model provides a fundamental fabric for solving a very large class of distributed optimizations. Once a problem can be written in the distributed ADMM form, it can be efficiently mapped and executed on the proposed test-chip, which we call OPTIMO. To introduce OPTIMO, we first present its architecture in the next section.

2.3 Combinatorial Optimization: Continuous-Time Dynamical System

Solving optimization problems has been widely used and brought invaluable impact to our world, however measuring the hardness of optimization is not that trivial and easy to quantize [37]. In [37], the authors proposed a way of mapping the k-SAT into a deterministic continuous-time dynamical system and show the relationship between transiently chaotic behavior and the appearance of optimization hardness. They show that beyond a constraint density threshold, not only the analog trajectories become transiently chaotic, but the boundaries between the basins of attraction of the solution clusters also become fractal,

and therefore the escape rate κ can be considered as a measure of hardness for the given problem.

2.3.1 k-SAT Algorithm

The goal of a k-SAT problem is to find an assignment to N Boolean variables $x_i \in 0, 1, i = 1, ..., N$, such that they satisfy a given propositional formula F. F in conjunctive normal form (CNF) is expressed as the conjunction of M clauses $C_m, m = 1, ..., M, i.e., F = \bigwedge_{m=1}^{M} C_m$, where each clause is formed by the disjunction of k literals, either are variables or their complements.

As described in [37], an analog variable $s_i = -1$ corresponding to x_i being FALSE ($x_i = 0$) and $s_i = 1$ to x_i being TRUE ($x_i = 1$). The formula $F = \bigwedge_{m=1}^{M} C_m$ can be encoded via a $M * N$ matrix $C = c_{m,i}$ with $c_{m,i} = 1$ when x_i appears in the clause C_m, $c_{m,i} = -1$ when ($\overline{x_i}$) appears in C_m, and $c_{m,i} = 0$ when neither x_i or ($\overline{x_i}$) appears in C_m. To every clause C_m, we associate an analog function $K_m(s) \in [0, 1]$ given by

$$K_m(s) = 2^{-k} \prod_{i=1}^{N} (1 - c_{m,i} s_i) \tag{2.10}$$

It is trivial to see that clause C_m is satisfied iff $K_m = 0$. If we further define a "potential energy" function

$$V(s, a) = \sum_{m=1}^{M} (a_m K_m^2) \tag{2.11}$$

where $a_m > 0$ are auxiliary variables, one can see that all the clauses are satisfied iff $V = 0$. Therefore the SAT problem can be reformulated as a search problem in s for a globla minima of V (since the condition $V \geq 0$ always applies).

As described in [20], while constant auxiliary variables may result in local minima of $V(s, a)$ and not able to find solution, the formal solution where amplifiers are required is not scalable and brings the unneglectable power overhead. Therefore, instead of using the

circuit which perfectly matches the formal solution, we applied the alternative circuit to realize auxiliary variables to trade with the power consumption which comes with $(1 - \epsilon_2 e^{-qt})$-type growth.

2.3.2 Previous Research

Since Boolean satisfiability is the quintessential constraints-satisfaction problem and lies at the basis of many circuit optimization [38], scheduling [39], and error-correction applications [40], many researches on finding an efficient solver has been conducted. Typically the problem is being tackled using conventional digital computing architectures, but because the lack of reflection of the distributed nature which is required for many applications, the typical ways are generally considered ill-suited and therefore many parallel analogue/digital hardware architectures have been recently proposed [21].

While some dynamical systems approach the problem with event-based computation focusing on physically realizability [21], there are some systems approaching with either voltage or current based computation [20]. Some may argue that the later approaches may violate the 'physical implementability' condition since the use of variables can grow without bounds as the system is searching for solutions [41, 42], such as the approach our research uses. However the main goal of this research is not to propose a powerful SAT solver, but instead using this SAT solver as a demonstration on silicon to manifest the relationship between optimization hardness and transient behavior, which we believe the later approaches are rather more suitable and straightforward.

When the physically realizability is focused, researchers had first used attractor networks, for example Hopfield networks [43], to solve some NP hard problems such as travelling salesman problem [10, 11]. However these networks often get stuck at locally optimal solutions, and therefore stochastic mechanisms were applied to overcome this problem [12, 13], with the penalty of explicit sources of noise. Furthermore, it is not trivial how to balance 'exploratory' versus 'greedy' search and thus it puts an additional overhead on the

physical implementation.

For SAT solvers, the currently best known deterministic sequential discrete algorithm that exploits the properties of the search space has a worst case complexity of $O(1.473^N)$ [14]. Some algorithms use heuristics and while they may perform well on some SAT formula classes, there are always formulas on which they either take exponentially long time to solve or get stuck indefinitely.

2.3.3 Approach

In the scope of this research, we consider designing analog circuits and implement using traditional CMOS technology for solving a representative NP-complete problem, the SAT problem, referred to as AC-SAT. AC-SAT is based on the deterministic CTDS and uses the form of coupled ordinary differential equations described in [37]. As mentioned above, this system finds SAT solutions in analog polynomial time with the expense of auxiliary variables growing exponentially when needed. Though this CTDS is an incomplete solver, it does minimize the number of unsatisfied clauses when there are no solution and becomes MaxSAT solver. As described in subsection 3.1.1, the overall design is programmable and modular, and thus it can readily solve any SAT problem of size equal or less than the hardware limitations and easily to be expanded to larger size with larger chip area.

We have validated our design through multiple stages, from high level simulation on the general purpose CPU, low level simulation through SPICE, all the way to the measurement on the fabricated chip. Through the measurement result, we demonstrate the relationship between optimization hardness as transient chaos and show that this architecture is highly scalable and configurable.

As shown in the Figure 2.2, the system flow contains: 1) Define the classic Boolean satisfiability problem. 2) Map the problem onto continuous dynamical system. 3) Configure the user-defined parameters. 4) Convert the configuration to the binary stream and scan into the hardware. 5) Reset the system and let it run until the system converges. 6) Retrieve

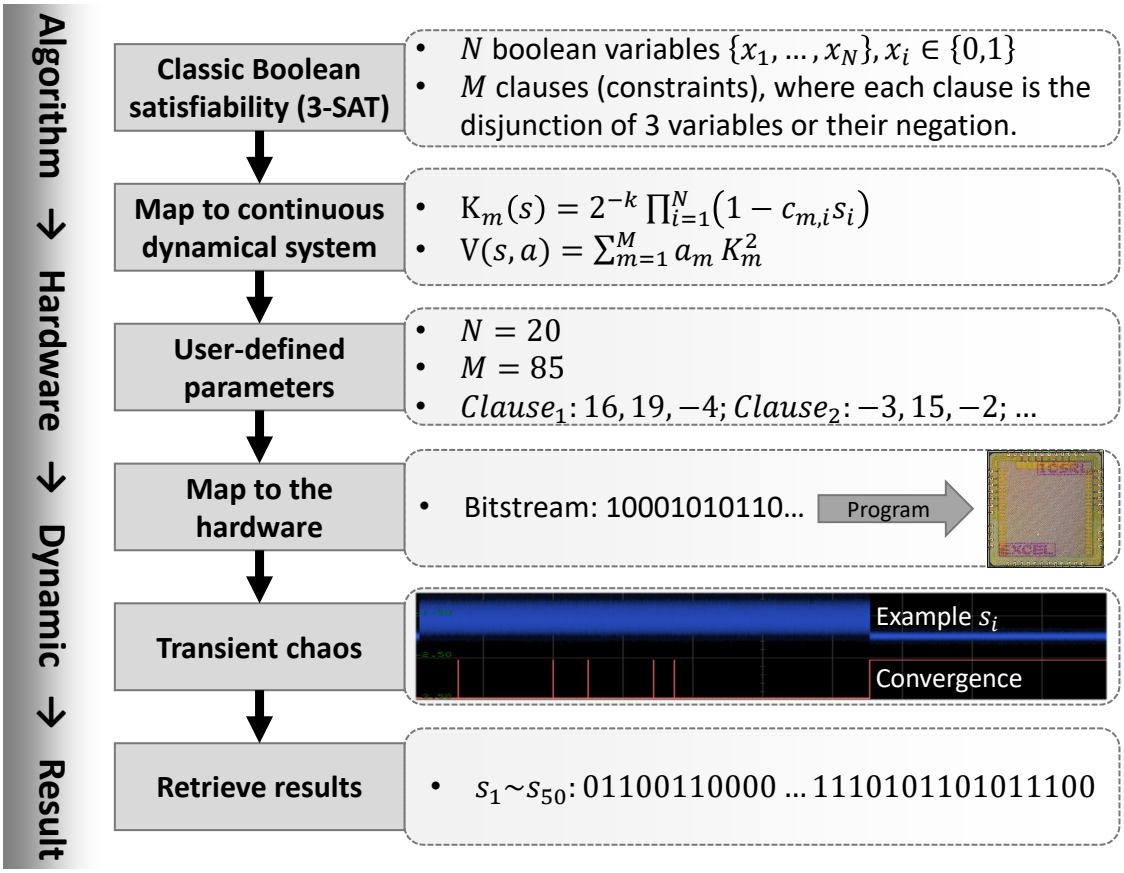


Figure 2.2: System flow.

the result by scanning out the variable values and verify the correctness.

CHAPTER 3

TECHNICAL APPROACH

3.1 FPGA Implementation

The design has been synthesized and implemented on a Xilinx Virtex-7 FPGA board. Table Table 3.1 lists the specifications of the Virtex-7 FPGA board. Also an emulator has been developed in software to help us program and debug the hardware, simulate large-scaled design and perform functional verification. Further energy efficiency can be achieved by designing an ASIC [44].

Model	Logic Cells	DSP Slices	Memory(Kb)	I/O Pins
XC7VX485T2	326,400	1,120	27,000	700

Table 3.1: Specifications of the Virtex-7 FPGA board

3.1.1 System Architecture

At the top level, the system can be easily reconfigured from a 2D to a 1D architecture by turning off communication channels in both the vertical and diagonal communication routers. To avoid complex clock tree synthesis and enable scaling without incurring the cost of routing global signals, the system does not have a global clock; instead, each individual core has its own local clock following a globally asynchronous and locally synchronous (GALS) topology. The individual core operates in two phases: computation phase and communication phase. To program the system before the start of an iteration, all the constants are scanned-in and written into the local memory of each core.

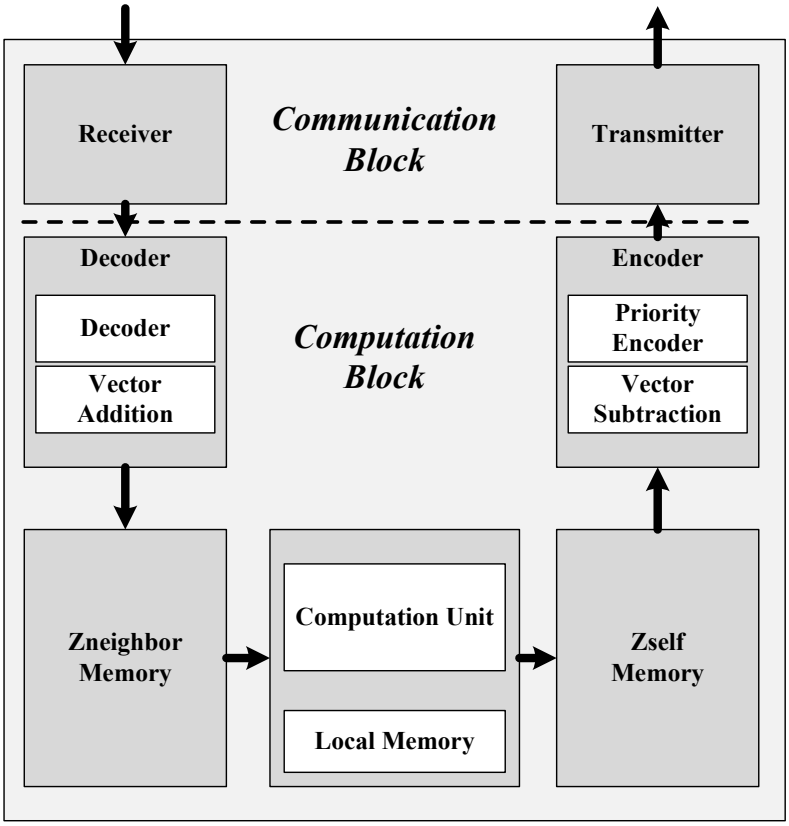


Figure 3.1: Core architecture.

3.1.2 Core Unit

Each core is locally synchronous and communicates with other cores through asynchronous mechanism. The structure of a core is shown in Figure 3.1. To take a start from the receiver, where a core receives the updated data from the neighbors and saved into $Z_{neighbor}$ memory, then processed the data in the computation unit, and at the end send to the neighbors through the transmitter.

3.1.3 Communication Unit

In order to reduce the number of wires between each pair of cores, we decided to encode and decode the information being transferred. In other words if we are using 32 bits integers, instead of transferring what the current coefficients are, we calculate the the deltas between the consecutive coefficients and figure out the location of the most significant bit

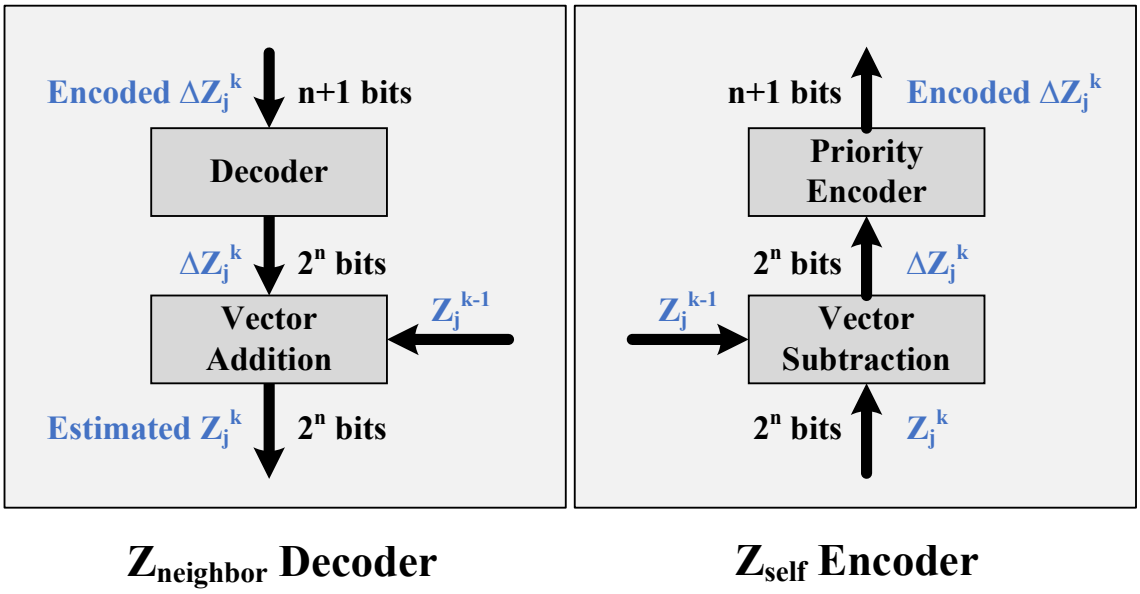


Figure 3.2: Encoder and Decoder

in the deltas. With this step integrated, instead of sending 32 bits integers, we only need to send $log(32) + 1 = 6$ bits. The block diagram of the encoder and decoder is shown in Figure 3.3.

During each iteration, once the computation is done, the core enters the communication phase. In this work, 4-phase handshake protocol has been used because of the reduced logical complexity and competitive power/area efficiency. Figure 3.3 shows the block-diagram of the communication unit. The implemented design uses Muller-C element to generate *Req* and *Ack* signal. Figure 3.3 also demonstrates the associated waveform diagram for the

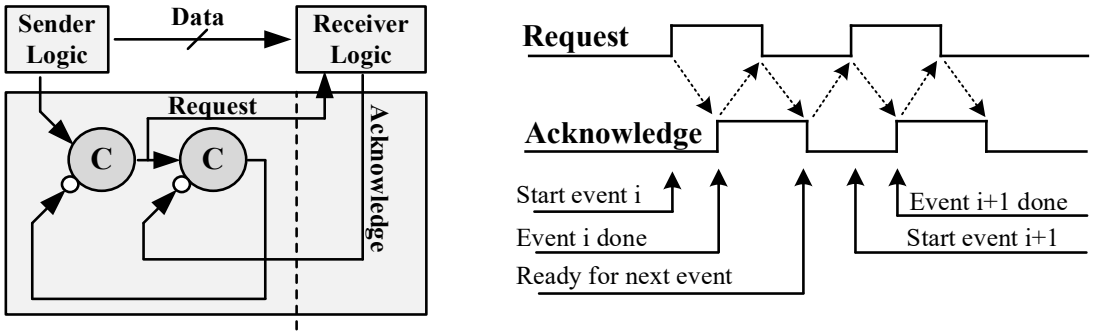


Figure 3.3: Schematic (left) and waveform (right) of the 4-phase hand shaking mechanism.

4-phase handshake protocol.

3.2 OPTIMO: Digital ASIC

3.2.1 System Architecture

Figure 3.4 illustrates the chip-architecture where 49 programmable 16b optimization processing units (OPU) are capable of (1) computing locally and iteratively and (2) transmitting/receiving data from the neighbors. The chip boundary has communication interfaces to the PCB that contain: (1) Scan ports (2) System control ports (3) Clock ports. High level program-code and data-sets are translated to instruction and data, scanned in to the chip

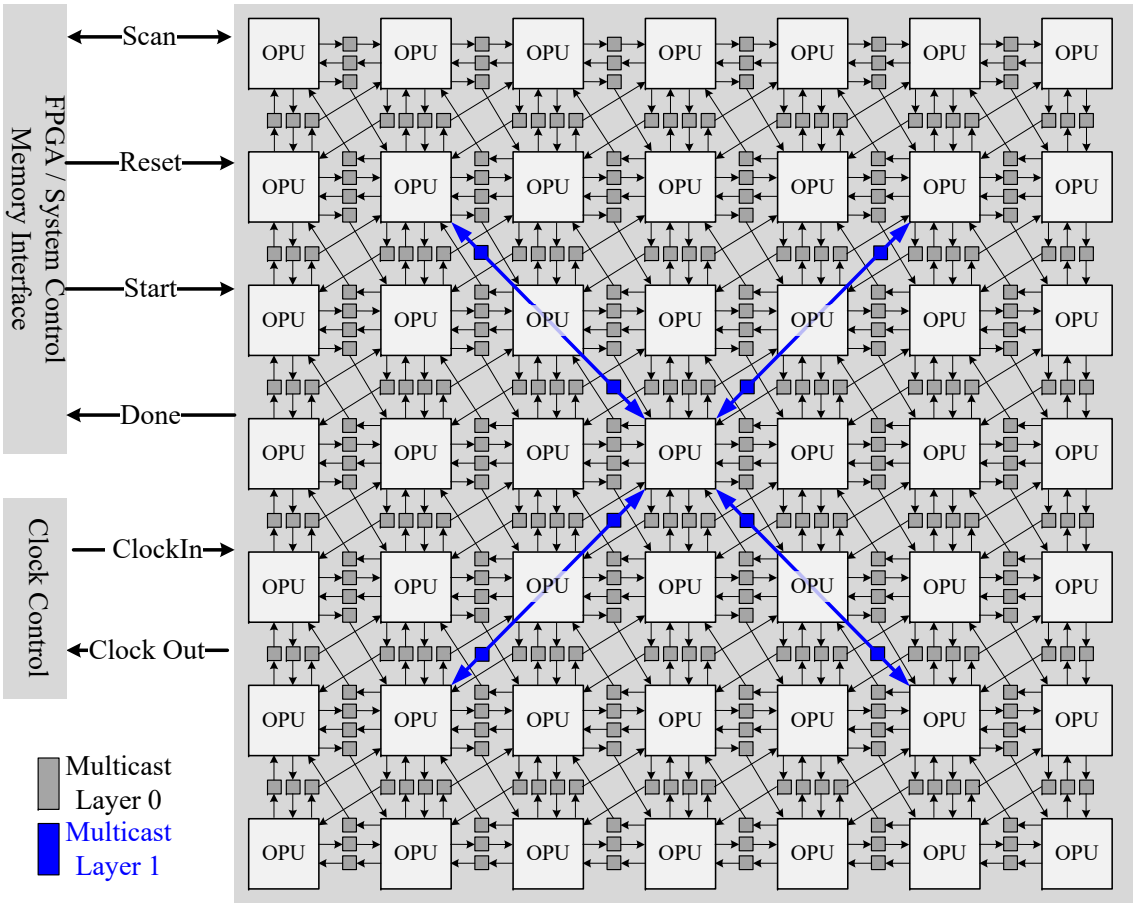


Figure 3.4: System Architecture showing the 49 OPUs and a 2-layer multi-cast on-chip network.

through scan ports and then executed. The control ports are used to start/reset the system and the clock ports are used to either provide the clock externally and/or monitor the system clock frequency. Convergence is declared either (1) after a fixed number of iterations, or (2) when the maximum cycle-to-cycle change of data in an OPU falls below a threshold. The system also contains two multicast layers, which will be described in section subsection 3.2.4.

The choice of 16b is driven by the data-set and the applications. For signal and image processing applications, that are of interest to us, the raw-data is 8b and we have determined that 16b precision yields the same results as floating point for the thousands of image and signal processing data-sets that we have analyzed. Further, ADMM is forgiving in terms of quantization error, and the system converges because of the iterative nature of the algorithm.

In the next section, The detail architecture of each OPU is described.

3.2.2 Optimization Processing Unit (OPU)

One of the important challenges for spatial-array architectures is scalability. In this design, we ensure that the IO pins for each OPU are placed in a symmetric fashion such that the OPUs can easily abut. Each OPU features (Figure 3.5): (1) one computation module consisting of a programmable digital signal processor (P-DSP), a scratchpad memory and control logic, (2) 2KB of instruction cache, (4) 4KB of data memory (for local data R/W), and (5) a transceiver module for the gather and scatter processes. Programming is supported via 32b instructions and each inter-OPU data-movement is supported on dedicated links. For this work, we mainly focused on how such architecture relates to iterative optimization problems; therefore we assume the weights and data can fit into each OPU. For more complex problems, the architecture can remain the same albeir with a more sophisticated NoC and higher bandwidth OPU to OPU bandwidth. Before data-transmission, a

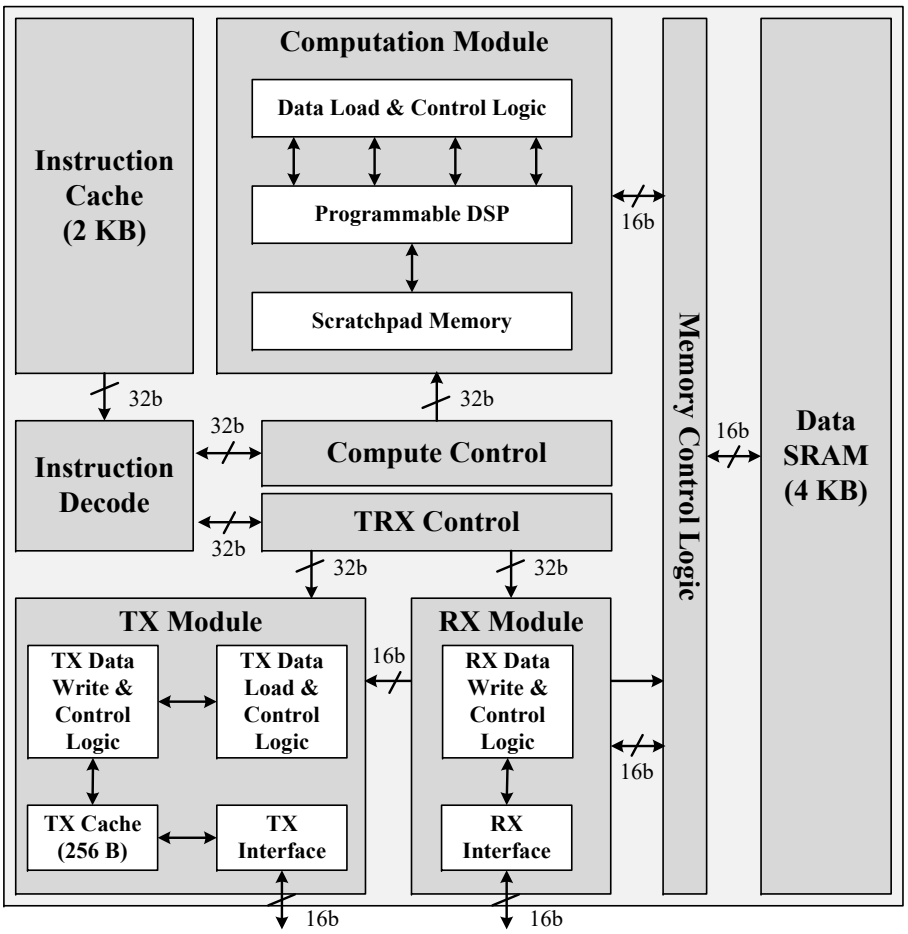


Figure 3.5: Architecture of the Optimization Processing Unit (OPU) showing the principal modules.

transmit buffer temporarily stores the data and it is flushed out at the end of the transmission. Received data is not buffered; instead the control logic directly writes the incoming traffic to the data cache thereby reducing both latency and energy. The design supports synchronous, mesochronous as well as asynchronous communication among OPUs with bidirectional FIFOs enabling fast and parallel data exchange across clock-crossing boundaries [45]. Table 3.2 illustrates the number of instructions supported by each module. Table 3.3 illustrates the commonly-used macro functions and the number of instructions per function as well as their usage in the six template algorithms.

Table 3.2: Number of custom instructions with a 32-b Instruction format and macro functions

Functional Blocks	Number of Instructions
Computation Controller	11
Communication Controller	8
Programmable DSP	768

Table 3.3: Programming support for each algorithm. (1*. Least Square; 2*. Lasso; 3*. Group Lasso; 4*. Elastic Net; 5*. Linear SVM; 6*. Distributed Averaging.)

Macro Functions	No. of Instr.	Algorithm Type					
		1*	2*	3*	4*	5*	6*
$L_1\ Norm$	3		✓	✓			
$(L_2\ Norm)^2$	3	✓	✓	✓	✓	✓	
$L_2\ Norm$	6		✓				
$L_{Inf}\ Norm$	3		✓				
$Shrinkage$	3		✓	✓			
$Multiply-Accumulate\ Operation$	4	✓	✓	✓	✓	✓	✓
$Hinge\ Loss$	3				✓		
$Distributed\ Averaging$	10	✓	✓	✓	✓	✓	

3.2.3 Programmable DSP

Figure 3.6 illustrates the principal components of the P-DSP, which consists of three pipeline stages in an architecture designed to maximize energy-efficiency. The first supports add/subtract (or bypass), the second stage supports multiply/divide (or bypass), and the final stage supports a class of fixed-function blocks as shown in Figure 3.6. The key fixed-function blocks are: Boolean Functions Processors, Shrinkage Function unit, a 16b arithmetic logic unit (ALU), a Hinge Function calculator and a Square Root function evaluator. Instruction level control of the pipeline and variable latency through the P-DSP is maintained via a program counter. The number of cycles required to execute an instruction will dynamically change depending on the type of instruction and the architecture configuration.

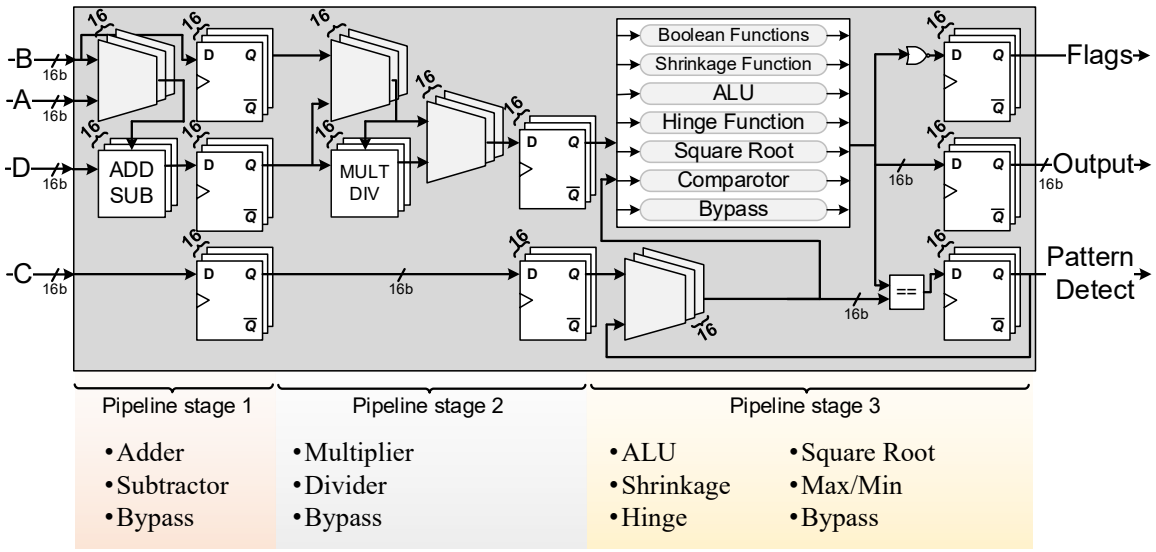


Figure 3.6: Programmable DSP Architecture showing a 3-stage pipeline.

3.2.4 On-chip Network

The OPUs indexed as (row, column) interact via a 2-layered multi-cast network with (1)*layer-0* establishing near-neighbor (neighborhood of 8) bi-directional connections and (2) *layer-1* connecting 4 cluster center OPUs i.e, (2,2), (6,2), (2,6) and (6,6) with the chip-center OPU i.e, (4,4). Depending on the algorithm and structure of the data, optimization algorithms require complex data-flow patterns where both near-neighbor (layer-0 connections) as well as global information (layer-0 and layer-1 connections) are used. The 48-OPUs (excluding the chip center) are divided into four clusters as shown, with the OPU in the center as shown in Figure 3.7. Global consensus is reached in each iteration via the following steps.

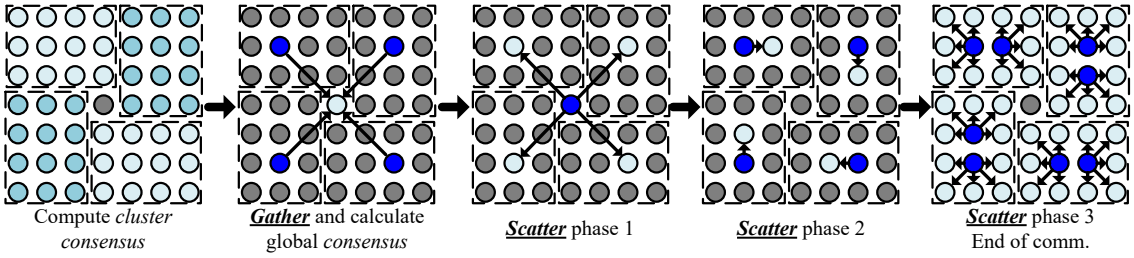


Figure 3.7: Gather and scatter processes of communication enabled by a hierarchical on-chip network result in fast convergence to a global consensus.

1. the four clusters reach cluster-level consensus (layer-0)

2. **gather** process where the chip-center obtains cluster-level consensus information from cluster centers (layer-1) and calculates the global data

3. **scatter** (step-1) process where the chip-center scatters the global data back to the cluster centers, and

4. a **final scatter** (step 2 and 3) process where the cluster-centers spreads the data across all the OPUs

Once these processes terminate the system will ready all the OPUs for the next iteration. The **scatter** and **gather** processes are intrinsic to distributed optmizations as the

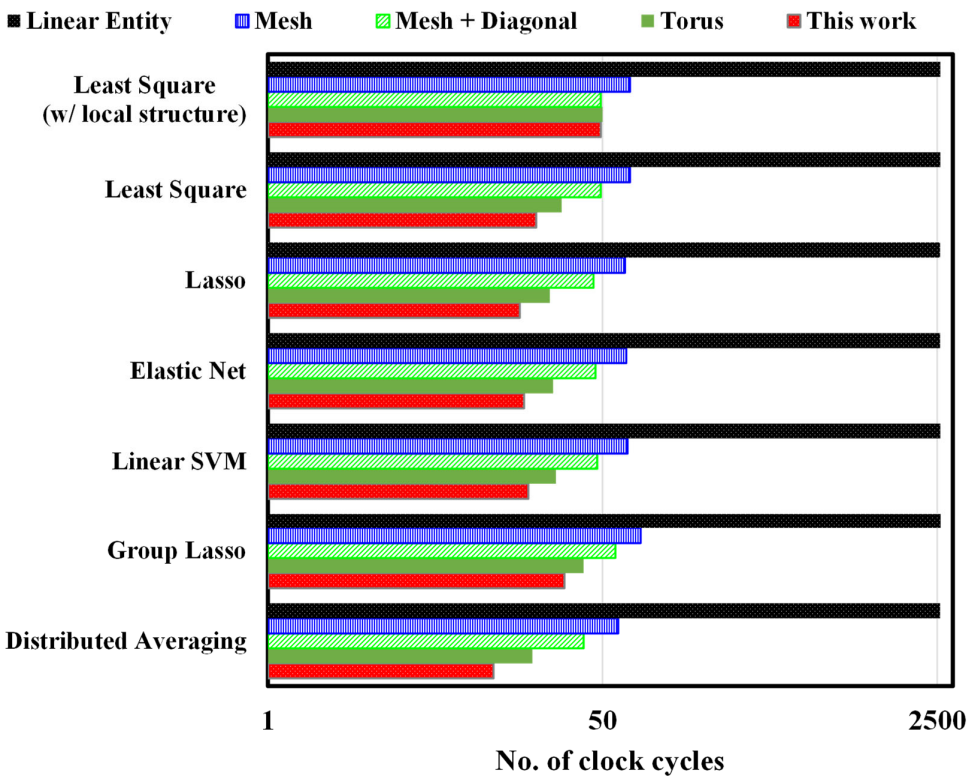


Figure 3.8: Time for convergence for template algorithms as a function of their connectivity with their neighbors.

Model	Linear Entity	Mesh	Mesh + Diagonal	Torus	This work
Complexity	Low	Low	Low	Medium	Medium
Convergence Speed	Low	Low-Medium	Medium	Medium-High	High

Table 3.4: Comparison between different network topology

system computes locally, distributes information globally and iterates to reach concensus. We compare the proposed hierarchical multi-cast network with networks that allow 4 or 6 connections to the neighbors – as is common in convolutional and deep neural networks [11]. It is intuitive to understand that instead of connections to all the 6 neighbors, consensus data can also be transmitted by just connecting to the four near neighbors (as found in Google's TPU). However, this comes at a cost of increases number of iterations. Architectural and network simulations of various optimization algorithms on more than 10000 random data-sets reveal a 29%-77% reduction in convergence time compared to the mesh topology (Figure 3.8). 'Linear Entity' represents the topology where all the OPUs are connected linearly and data has to hop from one to another through multiple OPUs and thus cause dramatic penalty. 'Mesh' represents the topology where each OPU is connected to horizontal and vertical neighboring OPUs. 'Mesh + Diagonal' builds on top of 'Mesh' and adds connections to the four extra diagonal OPUs. 'Torus' also builds on top of 'Mesh' and adds the wrap around connections on the very end OPUs. The comparison of complexity as well as the convergence speed between different topology is also shown in Table 3.4

3.2.5 Clocking

Clocking often becomes critical when scalability and power efficiency are considered. To overcome this issue, we implement two clocking options on the chip: (1) a single global clock (synchronous) either internal or external or (2) digital controlled oscillator (DCO) based clock per-OPU enabling asynchronous/mesochorous links. The DCO based local

clocks have external control via scan for fine-tuning. The single global clock option acts as the baseline for us to compare with per-OPU based clocking, where we show the comparison in section section 4.2. The system runs at full-capacity when all the OPUs are producing outputs at a fixed and equal rate – which requires synchronous communication. In such a scenario, no OPU has to wait for its neighbors to finish computation. However, per-OPU based clocking removes design constraints on fine grained control of clock-skew. As a compromise, mesochronous clocks running at identical frequencies but mismatched phases, maintain high throughput without requiring stringent skew requirements. To support mesochronous as well as asynchornous clocks per-OPU, the clock-crossing FIFO features a 64b buffer and operates on a 4-phase handshaking scheme, as shown in Figure 3.9.

3.2.6 Die Micrograph and Chip Characteristics

The test-chip is fabricated in TSMC 65nm GP CMOS process and occupies a total area of 12mm^2, as shown in Figure 3.10. It features 306.25KB of on-die memory distributed across 49-cores. The chip characteristics are shown in Table 3.5.

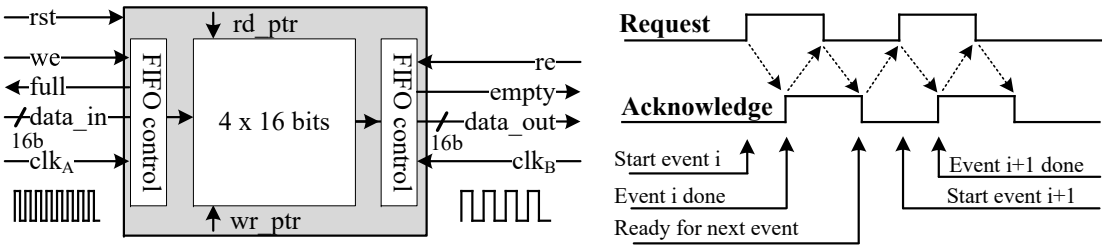


Figure 3.9: FIFO Architecture and the corresponding timing diagram.

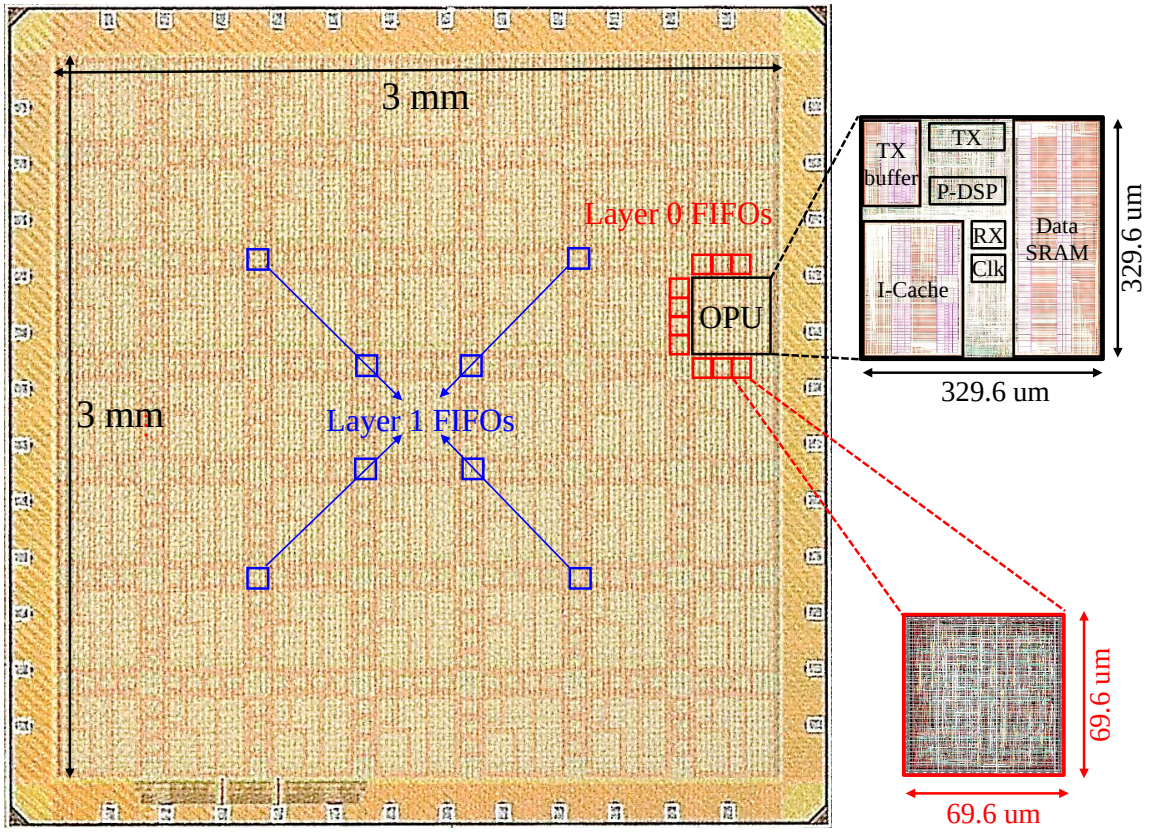


Figure 3.10: Die micrograph of OPTIMO

Table 3.5: OPTIMO Chip characteristics.

Technology	TSMC 65nm GP 1P9M
Chip Size	3.41 mm x 3.41 mm
Core Area	3 mm x 3 mm
Package	QFN6x6-48
Pin Count	48
Gate Count (logic only)	2725 kGates (NAND2)
On-Chip SRAM	306.25 KB
Number of OPUs	49
No. of pipeline stages in programmable DSP	3
Core Supply Voltage	0.5-1.2 V
IO Supply Voltage	2.5 V
Clock Rate	10-270 MHz
Network	Asynchronous & Mesochronous
Peak Energy Efficiency	279 GOPS/W
Arithmetic Precision	16-bit fixed-point

3.3 AC-SAT: Analog ASIC

3.3.1 System Architecture

In this section, we describe the high-level system architecture of AC-SAT solver. If the reader is not yet familiar with the definition of 3-SAT problems or how AC-SAT remaps the original 3-SAT problems onto the continuous-time dynamical system, we recommend to first finish the subsection 2.3.1 before continuing to read this section.

Although it is possible to implement the CTDS equations digitally, the required area and power of such hardware would be much higher for a system with decent performance. Therefore AC-SAT opts for an analog implementation with modular design and high configuration ability, leading to the need of more careful and elegant design considerations of the overall architecture as well as the physical implementation, which will be elaborated later in the section.

Figure 3.11 shows the high-level block diagram of AC-SAT. At system-level, it consists of four main components: (1) 50 S cells, which hold the dynamic and analog value of each variable before the system reaches the convergence. Inside each S cell, there is a loop formed by two back-to-back inverters with a control feedback signal FB_{on}. FB_{on} will be

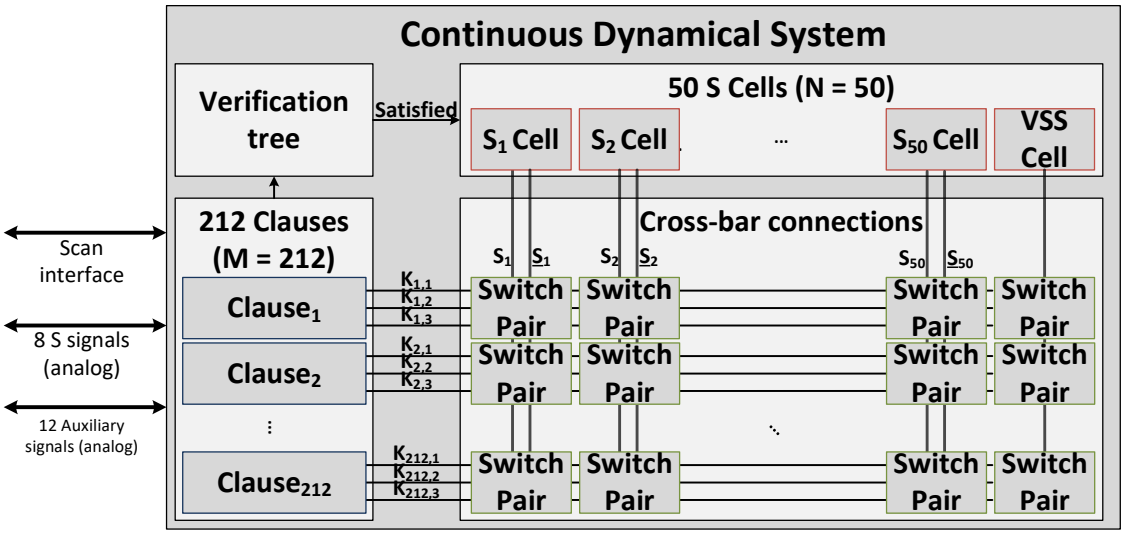


Figure 3.11: High-level block diagram of AC-SAT.

on when all the clauses are satisfied or will be turned off when we are initializing the whole system. When the system is being initialized, the values of S_n and ($\overline{S_n}$) will be set to S_{init} and ($\overline{S_{init}}$) through two transmission gates, respectively. (2) 50 x 212 cross-bar connection arrays, which control the connectivity between the clauses and the S cells. Notice that for the simplicity of illustration, we only show $K_{(m,i)}$ being connected through the switch pair, however in reality we need a pair of switches, one connecting S_n to $K_{(m,i)}$ and the other one connecting $\overline{S_n}$ to $\overline{K_{(m,i)}}$, which is also the reason we call it switch pair. (3) Verification tree which checks whether all clauses are satisfied and generate the feedback signal and send it back to the clauses as well as S cells. (4) 212 clauses, which act upon the current dynamic circumstance and respond to the connected S cells, how the continuous-time dynamic works will be walked through in the next section when we introduce an simple example of how to map the problem onto the hardware.

Each clause can be further decomposed into three parts: (1) Signal dynamic circuit (SDC), which implements the dynamics of variable signals and is responsible for interacting with S cells. (2) Auxiliary variable circuit (AVC), which implements the dynamics of auxiliary variables and is mainly for the purpose of preventing the system from falling into local minima. (3) Digital verification circuit (DVC), which simply checks whether the current clause has been satisfied. The detail of these three blocks will be introduced later when we talk about the detail of underlying circuit.

The given block diagram can solve any 3-SAT problem with up to 50 variables and 212 clauses, we chose such numbers based on the area limitation of the integrated chip. However, the modular design makes it very straightforward to scale up to desired number of variables and clauses, with the expense of some tuning on the transistor sizes as well as the load capacitors at the system level, which we will elaborate more in subsection 4.3.7 when we share some of the design challenges we faced.

3.3.2 Mapping to Hardware

To understand how the system operates, here we present a simplified example with only 10 S cells and the M/N ratio being 4.25, i.e. 10 variables, and 42 clauses.

As can be seen in the Figure 3.12, clause 1 is the disjunction of S_8, S_{10}, and $\overline{S_2}$, clause 2 is the disjunction of $\overline{S_2}$, S_8, and $\overline{S_1}$, so on and so forth. If we take a closer look at S_{10} specifically, it is trivial to see that inside the figure the clause 1, 5, 9, 41, and 42 are all constrained by either S_{10} or $\overline{S_{10}}$, therefore some of them will try to set S_{10} to 1 while some of them will try to set it to 0 before the system reaches the convergence obeying Kirchhoff's current law and based on the equations mentioned in [20]. Notice that the way clauses set the S variables is by either charge or discharge with current in the analog domain through the configurable connections in the cross-bar arrays.

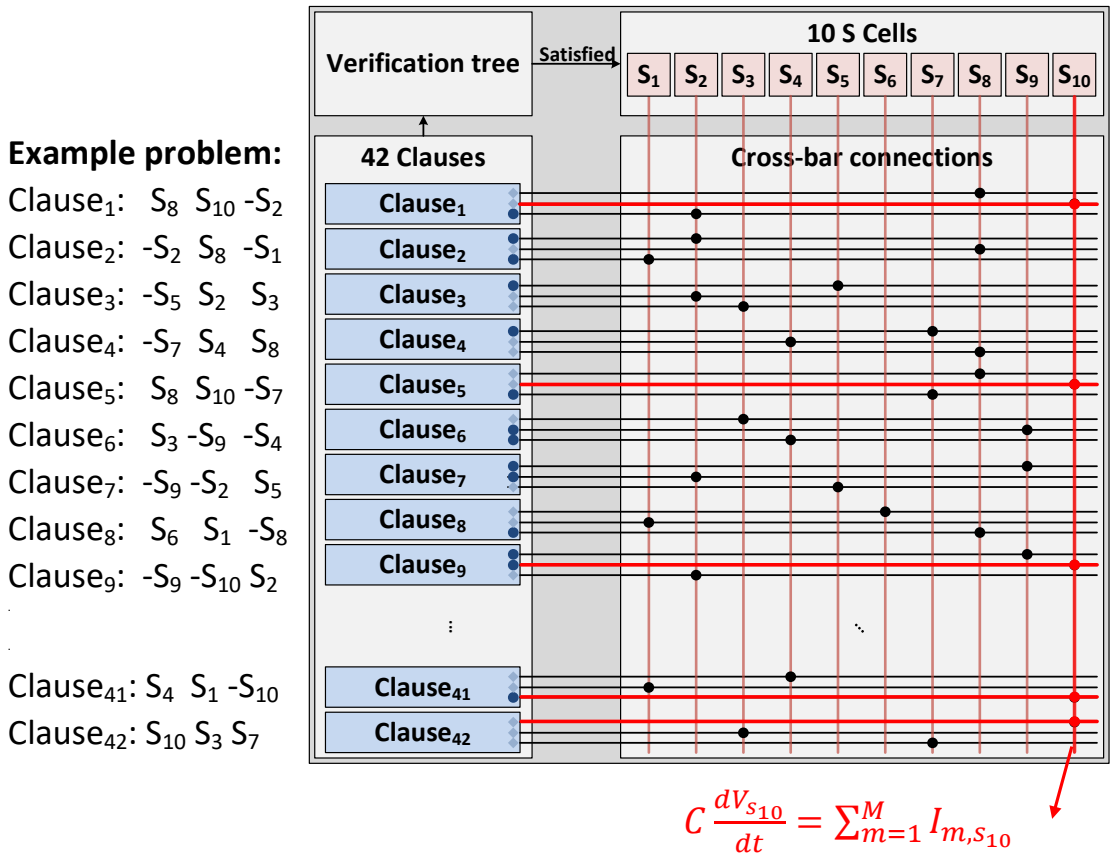


Figure 3.12: Example mapping from the simplified problem specification to the hardware.

As described in subsection 2.3.1, all the clauses are satisfied iff $V(s, a) = 0$. Thus, when all the clauses are satisfied, the signal $C_{satisfied}$ will be sent to all S cells as a notification to enable the feedback signal, FB_{on}, and settle down the solutions where each variable will be either pulled up to VDD or pulled down to GND.

3.3.3 Circuits of Blocks

As mentioned previously as well as shown in the Figure 3.13, each clause contains three parts: Signal dynamic circuit (SDC), Auxiliary variable circuit (AVC), and Digital verification circuit (DVC). Since [20] contains the detail of how the mathematical equations were mapped onto the circuit design, in this section we will focus on the intuitive explanation on how the circuit works from the perspective of current flow.

Signal dynamic circuit (SDC), which implements the dynamics of variable signals. As briefly mentioned, it may be easier to comprehend how this block works from the perspective of current charging and discharging. This block can be separated into two parts, where the first part relates to $Q_1 \sim Q_3$ and $K_1 \sim K_3$, and the second part relates to Q_1 and the auxiliary variable. Conceptually Q_i represents the digital value whether we want the variable S_i to be 1 or 0 defined in the constraint list and was pre-configured through the

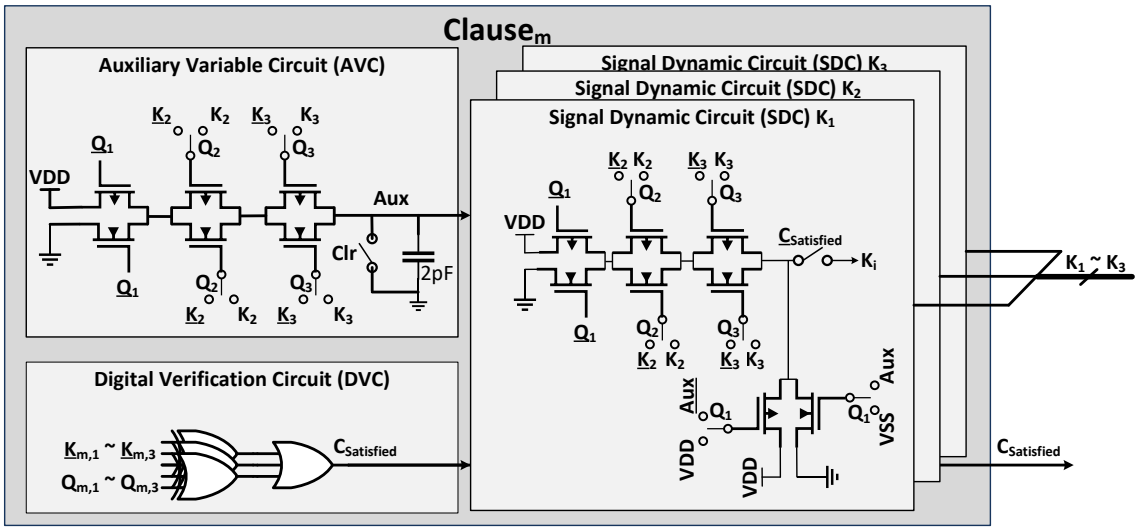


Figure 3.13: Detail design of a clause.

scan chain onto the hardware; K_i represent the analog value coming from the cross-bar connections. Being said, for the first part, the combination of Q_i and K_i controls whether we want to increase or decrease the current flowing from the SDC to the S cells through the switch pairs or the other way around. The same logic applies to the specialized transmission gate controlled by the auxiliary variable. One may find it easier to view the first part of SDC as the main operator and the second part as the safety option to prevent system from getting stuck at local minima. Once the clause is satisfied, the digital verification circuit (DVC) would disconnect the SDC from the cross-bar connections to prevent it from over-controlling the variable after the system enters random search phase in the cases where auxiliary variable has reached the physical limitation, in this case the auxiliary variable is bounded by VDD.

Auxiliary variable circuit (AVC), which implements the dynamics of auxiliary variables, works in the way that if none of the three variables in the current clause is satisfied, the auxiliary variable will keep increasing until it is bounded by VDD. This auxiliary variable will be passed to the SDC as the control signal inside SDC, as mentioned in the previous paragraph.

Digital verification circuit (DVC), which simply checks whether the current clause has been satisfied, is the last main block in each clause. By the definition of 3-SAT, as long as one pair of Q_i and K_i matches, this clause becomes satisfied and therefore output a $C_{satisfied}$ signal to the SDC as well as the verification tree at the higher hierarchy in order for the system to reach concensus.

The spin cell and the switch pair are fairly simpler. As shown in Figure 3.14, inside each spin cell it contains a back-to-back inverter loop with feedback controlled switch, the initial values can also be configured externally. Once the system is satisfied, the feedback switch would be closed and the spin variable would settle to either 1 or 0. For each switch pair, as S_n and $\overline{S_n}$ passing through, there are three independent control signals to set each pair of switches to be on or off .

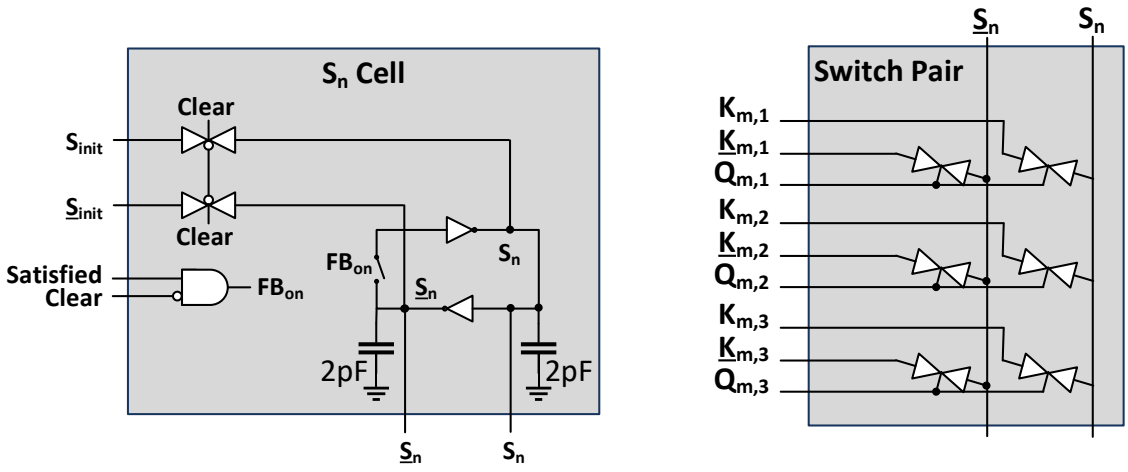


Figure 3.14: Detail design of a spin cell and a switch pair.

3.3.4 Transient Example

In this section we want demonstrate how the system runs in real time and the result is captured on Keysight MSOX4154A oscilloscope, which supports concurrently up to four analog inputs and sixteen digital inputs. Notice that even though we have maximum of 50 variables on the integrated chip, due to the limitation of the chip packaging and testing equipment, we are only able to probe 3 of them at one time. Since all the variables are equal, in order to have the ability to probe any variables freely, during the compilation process the system is capable of remapping the variables so the ones we are interested are connected to the pins which are exposure on the chip pins.

Figure 3.15 shows a general transient behavior of any problem specification. For the signals inside the figure, we have the waveform of S_1, S_2, S_3, the convergence signal, and the reset signal (Active-high) from the top to the bottom, respectively. After we scan in the configuration through the system scan chain interface, we first pull high the reset signal to clear the values from the previous run and reset each S cell to the initial conditions, which in this particular example the initial conditions are all zeros. After we release the reset signal, the system starts operating on its own and follows the continuous-time dynamic. Once the system reaches the convergence, we can see from the figure that the convergence

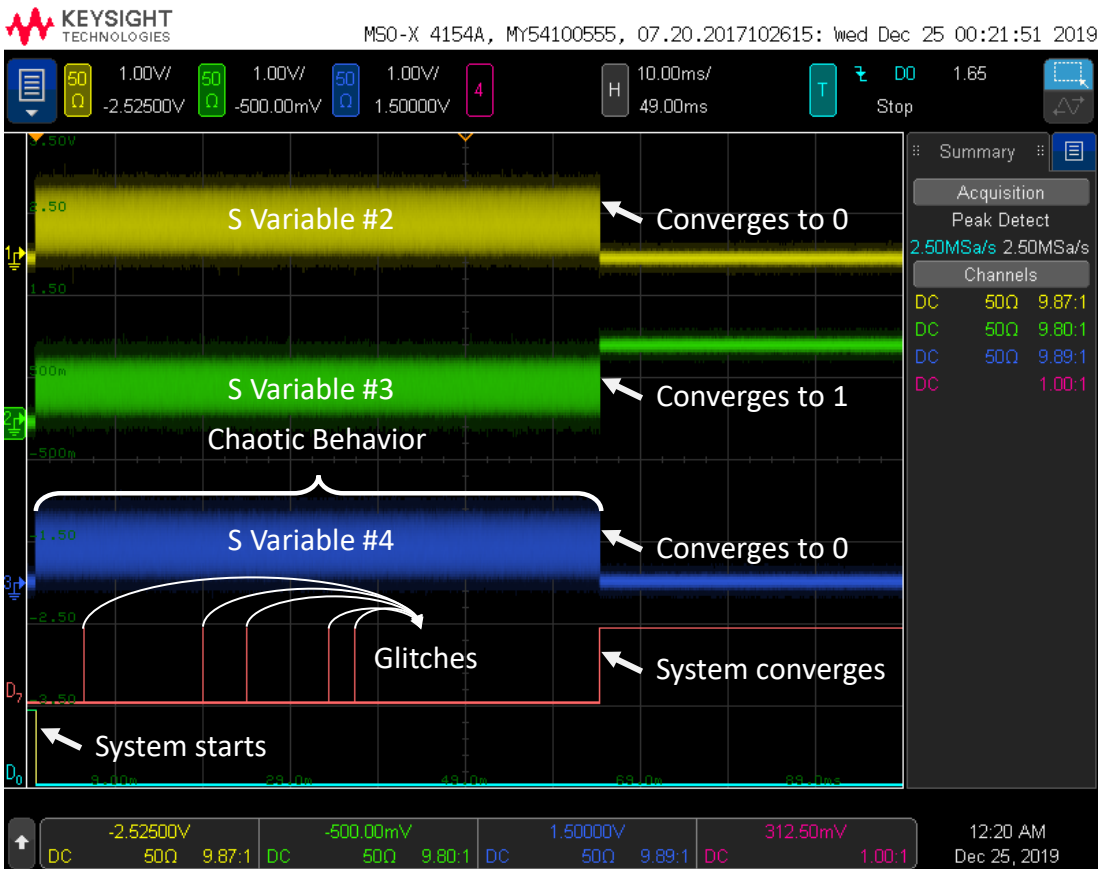


Figure 3.15: Transient Example.

signal becomes one, and all the variables are converged to either zero or one.

3.3.5 Die Micrograph and Chip Characteristics

The test-chip is fabricated in TSMC 65nm GP CMOS process and occupies a total area of 6.25 mm^2, as shown in Figure 3.16. It features 50 variables and 212 clauses with configurable and scalable architecture. The chip characteristics are shown in Table 3.6.

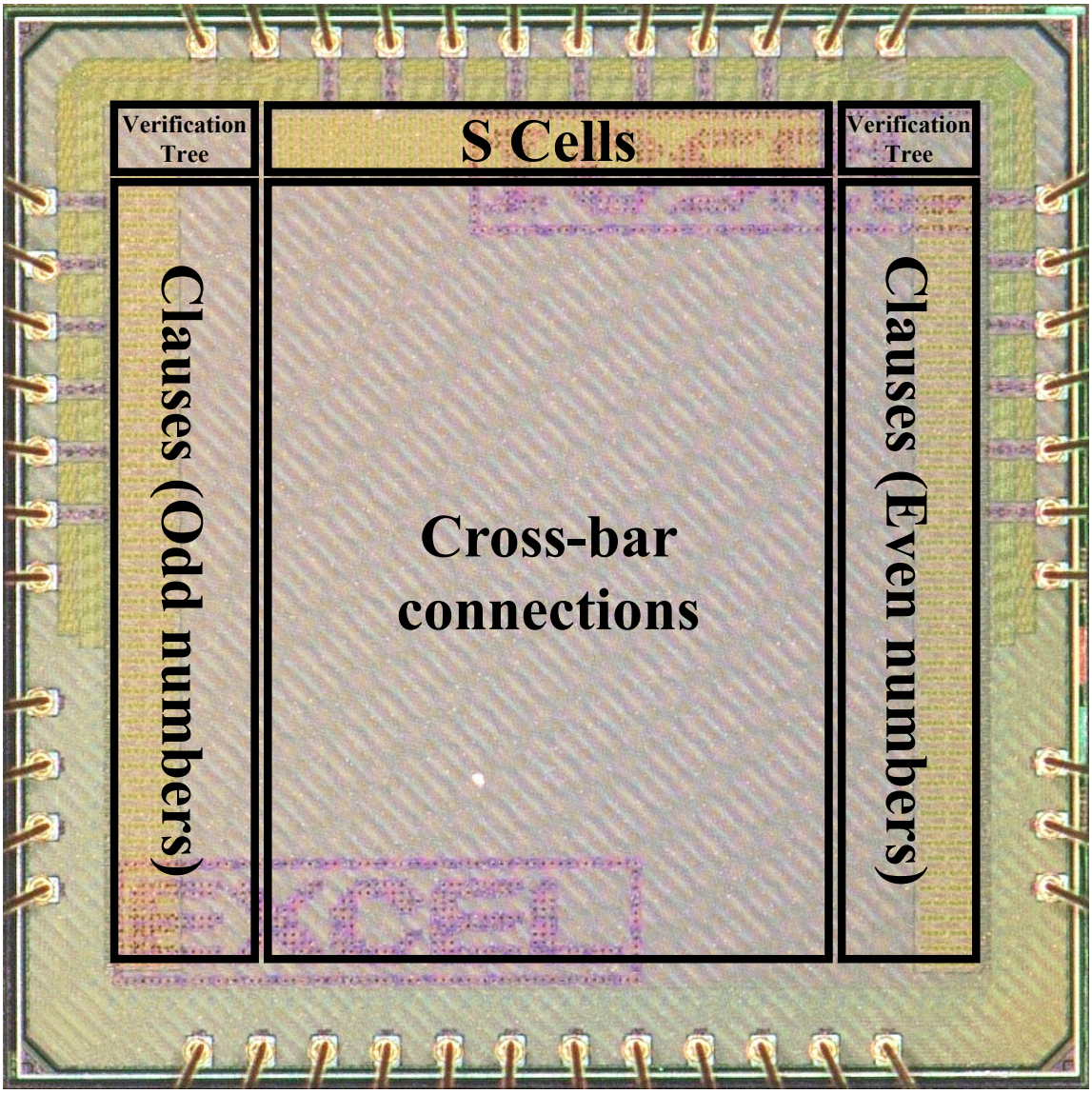


Figure 3.16: Die micrograph of AC-SAT

Table 3.6: AC-SAT Chip characteristics.

Technology	TSMC 65nm GP 1P9M
Chip Size	2.5 mm x 2.5 mm
Core Area	2.1 mm x 2.1 mm
Pin Count	45
Core Supply Voltage	0.5-1.2 V
IO Supply Voltage	3.3 V
Number of Variables	50
Number of Clauses	212

CHAPTER 4

RESULTS

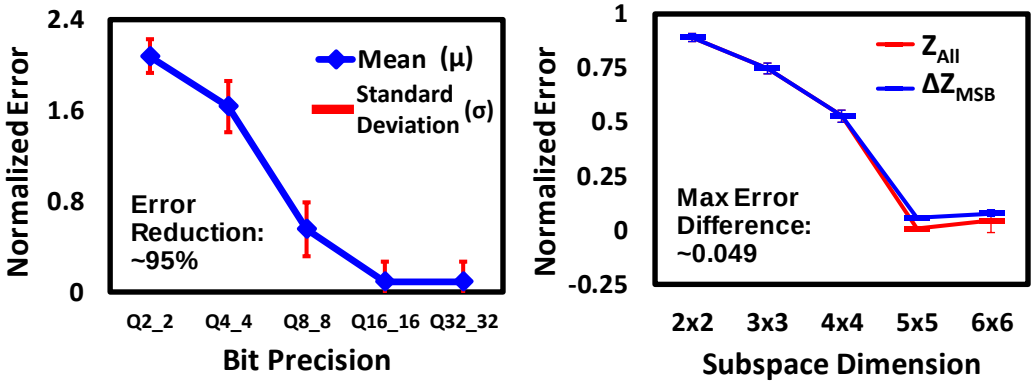
4.1 Continuous Optimization: Non-Uniform Sampling and Signal Reconstruction

Various key aspects of the design are studied and discussed in the following subsections.

4.1.1 Accuracy Analysis

Data Resolution

The number of bit used to represent the variables is important not only because it affects the estimation, but also because it affects the hardware architecture. Here we use Q-format to represent numbers. As can be seen from Figure 4.1a, the normalized error for signal reconstruction decreases dramatically after 16 bits and saturates when the bit precision reaches 32 bits.



(a) (b)

Figure 4.1: (a) Measured static error versus different value of bit precision. (b) Measured static error versus different size of subspace dimensions.

Subspace dimension

Figure 4.1b shows the accuracy when the number of subspace dimensions varies. Here, the original signal is synthesized under the subspace dimension of 5x5, and an interesting trend is observed. In the beginning, as subspace dimensions increase, the error reduces. However, we note a reversal of this trend in the error when the subspace dimension is larger than 5x5. This can be attributed to the over-fitting noise.

4.1.2 Design Scalability

Power Consumption

Figure 4.2a and Figure 4.2b show a linear increase in power with scaling for both 1-D and 2-D architectures. This shows an excellent scaling path for larger data-sizes. Also, as mentioned before the ΔZ_{MSB} encoding method reduces the size of the interconnects and hence the routing overhead. From measurements on the FPGA platform we note an average of 44% reduction in power consumption for both 1D and 2D architectures.

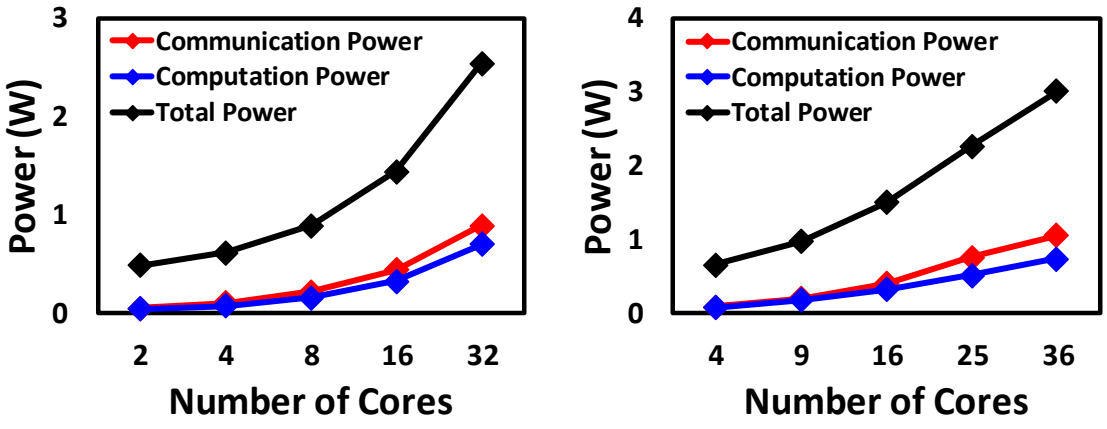
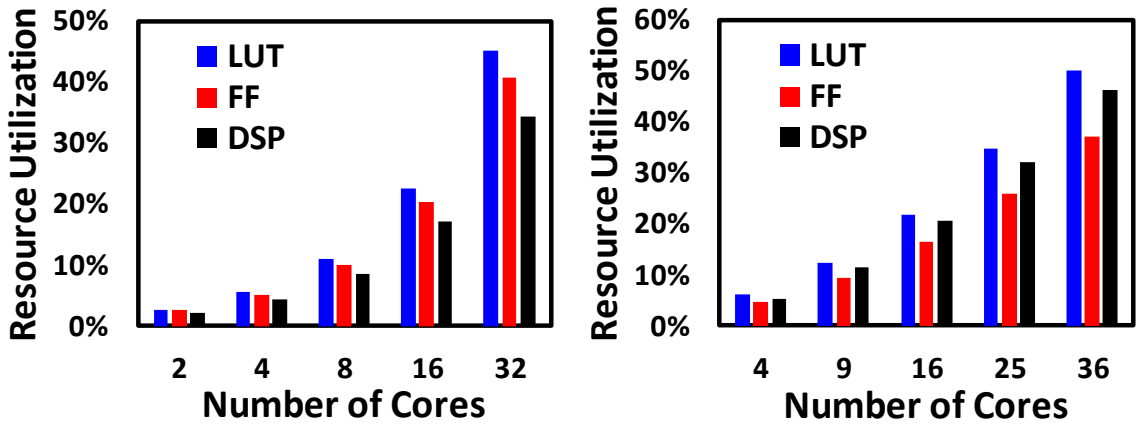


Figure 4.2: Measured power consumption of system on FPGA. (a) 1D case. (b) 2D case.



(a) (b)

Figure 4.3: Post-synthesis resource utilization on the FPGA platform. (a) 1D case. (b) 2D case.

Resource Utilization

The amount of resources which the system requires also increases almost linearly as shown in Figure 4.3. Here, it is important to point out that the major bottleneck for scaling is the interconnect routing inside the FPGA.

The measured results show that the modular and scalable design with the GALS architecture introduces high degree of parallelism and allows large problems with large data-sets to be mapped into the architecture. This makes co-design of iterative algorithms and systolic-architectures a powerful paradigm for solving distributed optimizations over large data-sets.

4.1.3 Design Challenges

At this fairly early stage of Ph.D program, the major design challenges on FPGA design mostly come from technical aspects, such as whether the code is synthesizable, whether we are running out of the FPGA resource, whether we are configuring the FPGA properly ... etc. Because of the things mentioned, I personally treat FPGA design as a very useful and practical pre-work before diving into silicon design. As I start recalling the challenges I was facing back then, one specific challenge is dealing with “for loop” in verilog as sometimes

it is synthesizable, sometimes it is not, and sometimes it synthesizes to something that we were not expecting depending how the for loop is constructed. One good example would be doing matrix multiplication in hardware. Coming from a programming background, I was so used to thinking in high level language scope therefore passing a whole matrix to a module seemed nothing wrong at all, not to mention fully parallelizing the whole matrix-matrix multiplication process. However there are so many downside doing this, for example we need to buffer two whole matrices and the bottleneck would fall onto the process of reading from the memory. Another big issue would be resource utilization ratio, although FPGAs have become more and more resourceful, none of them are barely this resourceful to provide this many of ALU or digital signal processing (DSP) units. At the end of the day, I had to think in the hardware method as breaking down the operations and fulfill the target using many basic elements such as counters, registers ... etc.

4.2 Continuous Optimization: Alternating Direction Method of Multipliers (ADMM)

The test-chip is packaged in a QFN package and integrated on a PCB with the necessary passives and connectors on board. It is programmed via serial scan through an external FPGA. Before the system starts, the instructions for each OPU are scanned in and followed by a 'start' signal, the FPGA then waits for the 'done' signal of the system. Measured electrical performance and algorithm level benchmarking are presented here.

4.2.1 Maximum Frequency and Power Consumption

Figure 4.4a illustrates the measured power-performance trade-off showing a peak F_{MAX} (in a synchronous setting) of 270 MHz (at 0.5V) and an operation down in 0.5V (with F_{MAX} = 10 MHz).

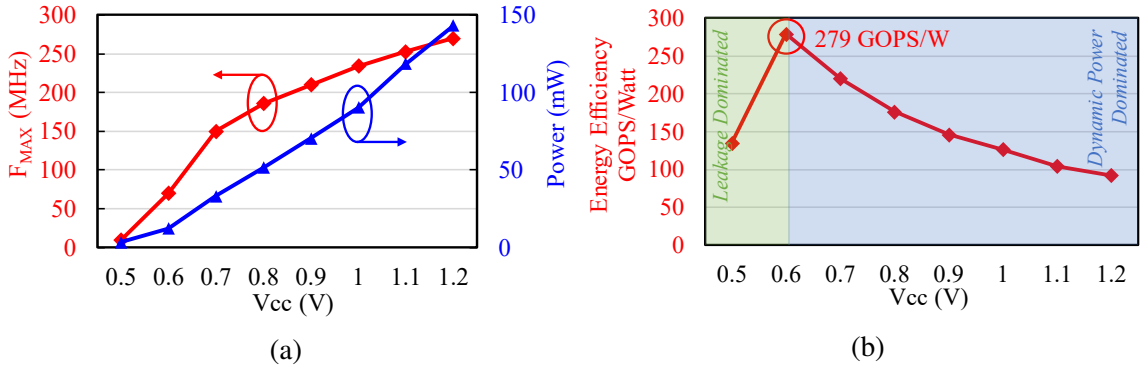


Figure 4.4: (a) Measured maximum frequency and power consumption. (b) Measured energy efficiency.

4.2.2 Energy Efficiency

Figure 4.4b shows as the operating voltage is reduced, the dynamic energy scales as V^2 whereas the time to complete the computation increases, thereby increasing active leakage power. The peak energy-efficiency, considering both dynamic and leakage power, is measured at 0.6V where we note a peak-efficiency of 0.279TOPS/W. Below 0.6V, the design is leakage dominated owing to the large (306.25 KB) on-die static random-access memory (SRAM). It should also be noted that an operation here represents execution of a single pipeline-stage of the P-DSP and is computationally more demanding than MAC operations which are often considered as a benchmark for signal processing or neural network accelerators. Per-OPU DCO-based clocking reduces the overhead of routing a global clock. We measure 2.7%-7.75% power savings compared to a fully synchronous global clocking strategy. This is measured at iso-performance by ensuring that the system throughput for both the synchronous and asynchornous/mesochoronous designs over a long measurement window is identical.

4.2.3 Power Breakdown

The power-breakdown among computation, communication and storage at 0.6 V and 1.0 V is shown in Figure 4.5. We see that the power consumed by all three components are almost

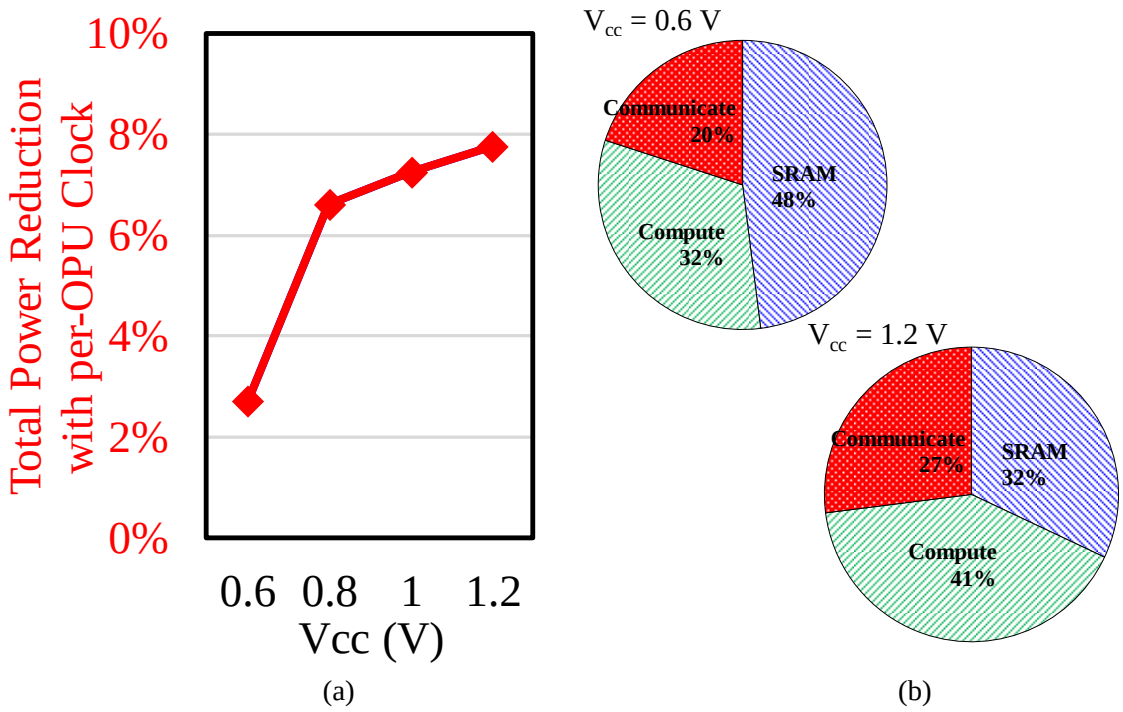


Figure 4.5: (a) Total power reduction for asynchronous design (per-OPU clocking) compared to a purely sequential design, (b)Measured break-down of power consumption.

equal at 1.0V and the system is dominated by SRAM power (mostly leakage) at 0.6V. Thus distributed optimization, as presented here, shows an interesting class of algorithms where computation, communication and storage are almost equally important, in terms of power consumption.

4.2.4 Time and Power Benchmark

We use the hardware prototype to execute template algorithms across multiple data-sets and plot the time-to-compute and energy at 0.6 V (Figure 4.6a and Figure 4.6b). The data-sets are generated at random and *MATLAB* based simulations are used to ensure correct functionality and convergence. The error-bars indicate the range of energy and performance required for different data values in the data-sets. We also note that group LASSO and linear SVM require the most number of iterations and energy – which is as expected, given the complexity of these algorithms. Although we demonstrate the capability of this near-

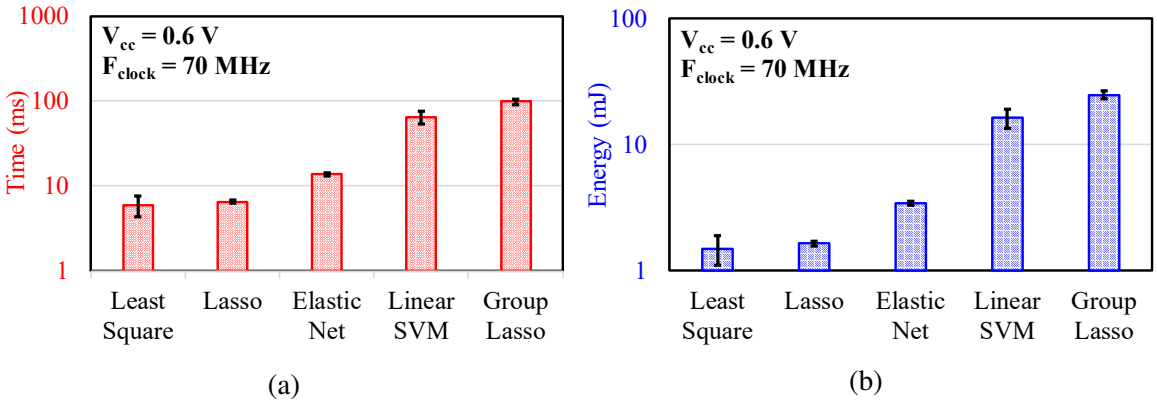


Figure 4.6: (a) Measured algorithm-level benchmarking showing the time to compute for six template algorithms. The errors bars show different problem instances that were characterized. (b) Measured algorithm-level benchmarking showing the energy to compute for six template algorithms. The errors bars show different problem instances that were characterized.

memory spatial-architecture in solving distributed optimizations, the proposed hardware and programming model can also support a variety of other array processing tasks as well, including inference in deep and convolutional neural networks. The on-chip network and the programmable P-DSPs allow flexibility to map such neural network based computing models, albeit with less energy-efficiency that fixed-function accelerators.

4.2.5 Design Challenges

In order to make the architecture of the system very flexible and modular, the pin assignments of each OPU has been thoroughly designed. This allows more cores to abut and the system can be used to solve larger problems. Communication occurs only to neighbors that can be easily extended to more number of cores. For cluster communication, one may envision another layer in the communication network that can provide faster global data movement. Owing to the modularity of the design, we believe that the system can be easily scaled to hundreds or thousands of cores as needed. In the current chip we have also provided asynchronous communication between cores that aids in scalability. Further the physical design is also carefully crafted such that each OPU is symmetric with ports that allow they to abut easily as shown in Figure 4.7.

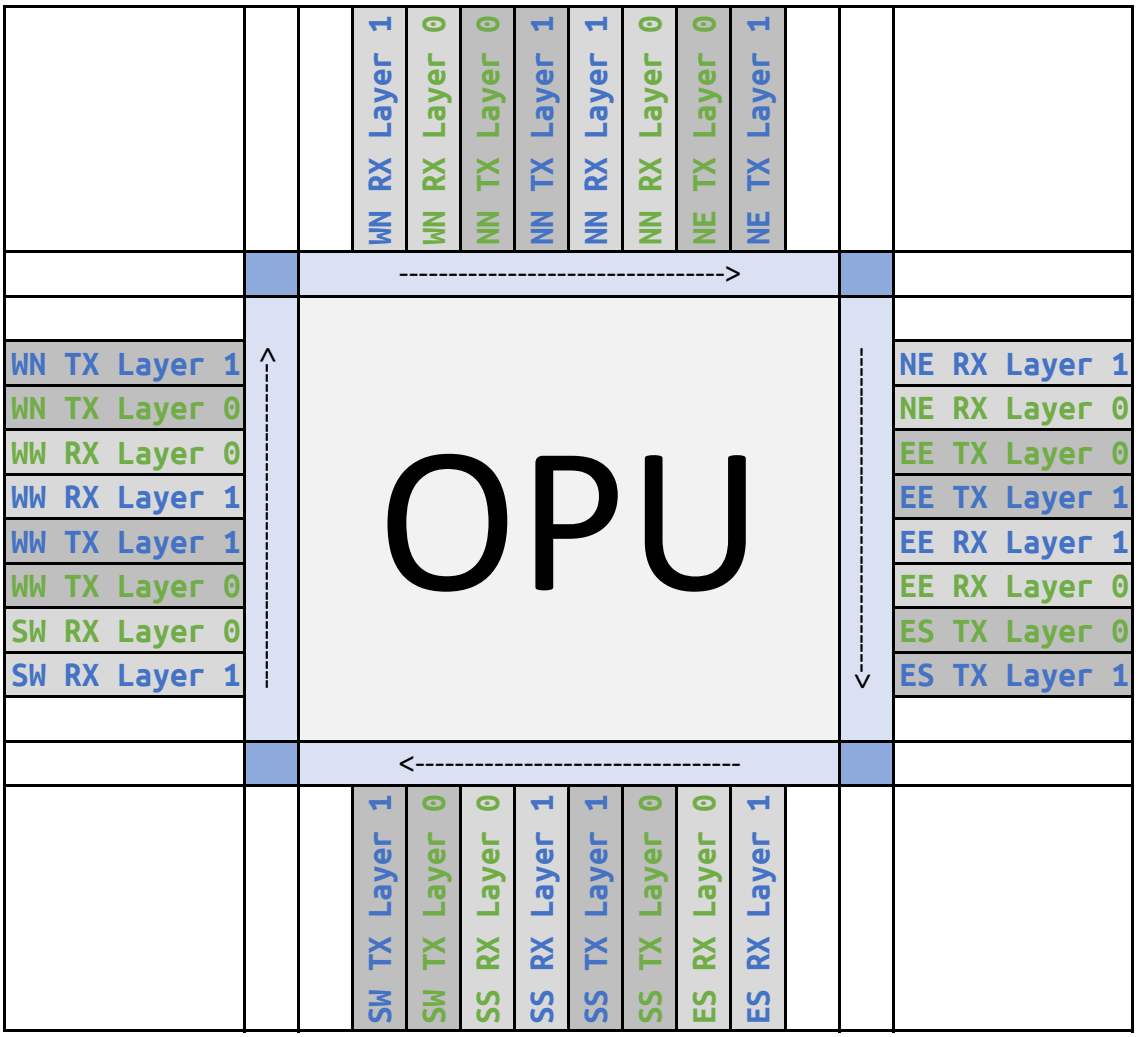


Figure 4.7: Pin Assignments for each OPU.

4.3 Combinatorial Optimization: Continuous-Time Dynamical System

4.3.1 Simulation on CPU

To show the strength of the circuit, we implemented the CTDS solver in C code running on the Intel I5 processor and compare it with the circuit, as shown in the Figure 4.8. We run 1000 problem per size (N) for the circuit solver and 20000 per size for the C code solver. Note that because it takes a lot longer to run fine grained circuit simulation, therefore it limits the number of problems we were able to run on the circuit solver. In the Figure 4.8, we can see that not only the trend of both solvers are similar, as the continuous-time dynamical

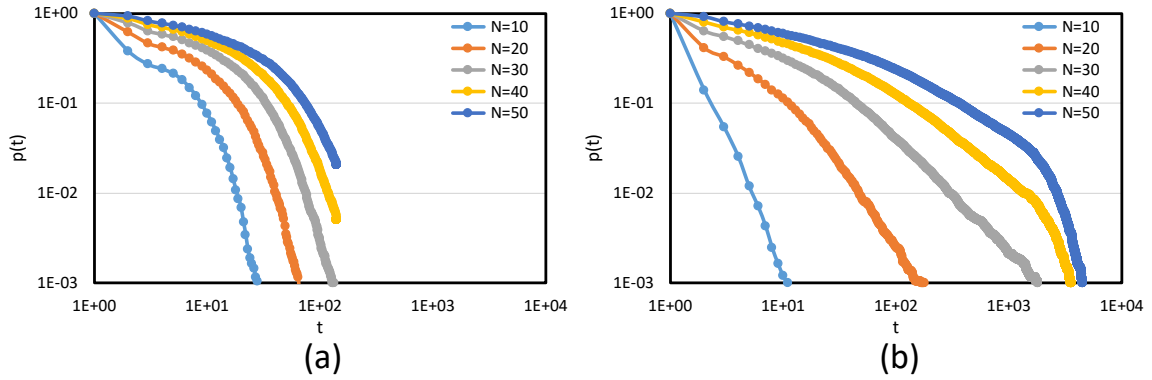


Figure 4.8: The fraction of problems p(t) not yet solved by continuous time t for 3-SAT at N=10, 20, 30, 40, 50 (colours).(a) Solver implemented in the circuit (AC-SAT). (b) Solver implemented in C.

solver, circuit version is much faster than digital version.

4.3.2 3D Plot of Dynamic Behavior

As can be expected, even with the same ratio, i.e. same number of clauses and variables, some problems are intrinsically more difficult than the others, thus we are particularly interested in how the continuous-time dynamic behaves visually in the 3D space and how they differ. In order to do so we conducted many measurements on the hardest benchmark k-SAT and sort the transient data according to its convergence time. We then pick four different results with different level of convergence time and plot them in the 3D space, as shown in the Figure 4.9. For each 3D figure, the axis are the values of S_1, S_2, S_3, and the color represent the normalized time stamp throughout the transient response. Though we will go into each case more deeply and show the corresponding transient behavior in the next paragraph, here even with only the 3D plots we can see clearly that while for easy problems the system quickly transition from the initial condition to the convergence point without traversing around the solution space much, for medium-level problems the system searches in the space for a while at the early stage before reaching the convergence point. Furthermore, for some very difficult problems the system takes a large amount of

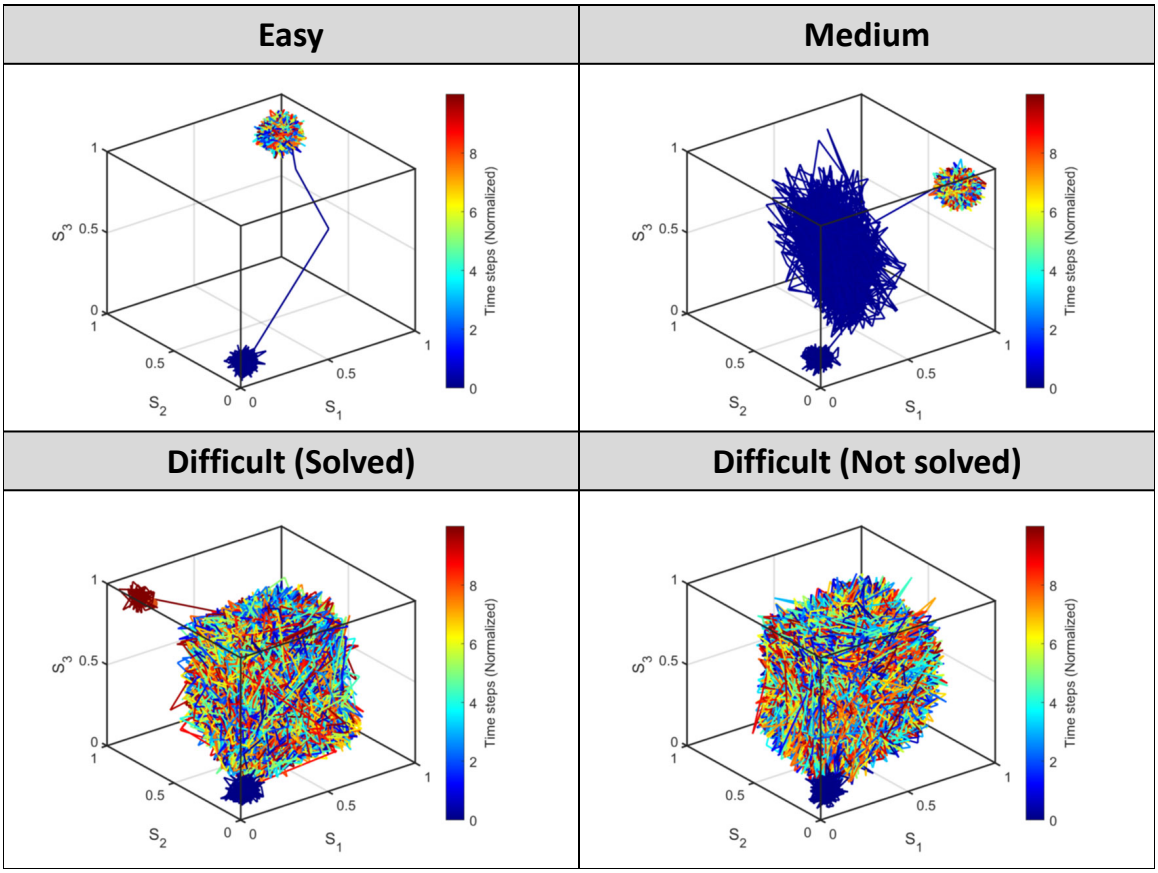


Figure 4.9: 3D Plot of Dynamic Behavior.

time searching through the space but eventually reaches the convergence point. However, for some extreme cases the system does not reach the convergence, but instead randomly search through the space in chaotic once the system reaches its physical limitation.

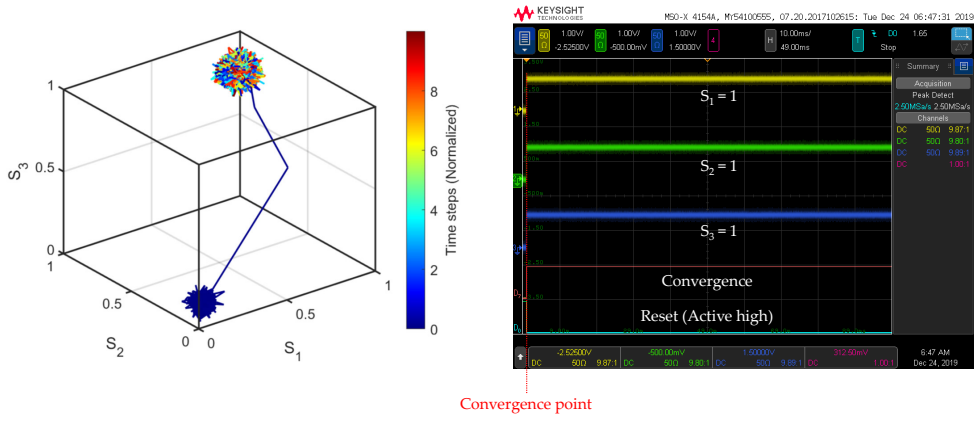


Figure 4.10: Easy case.

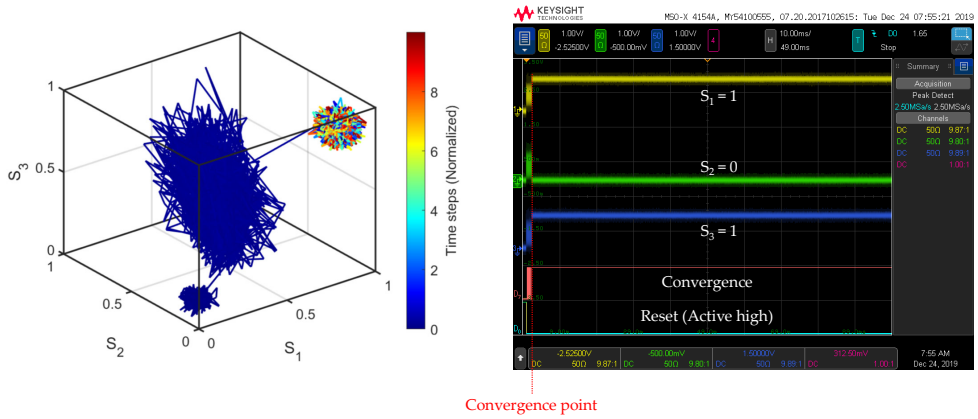


Figure 4.11: Medium case.

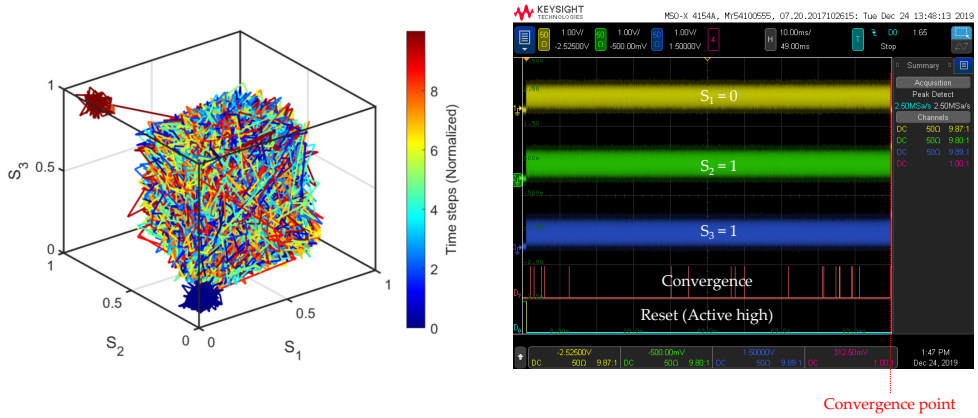


Figure 4.12: Hard but solvable case.

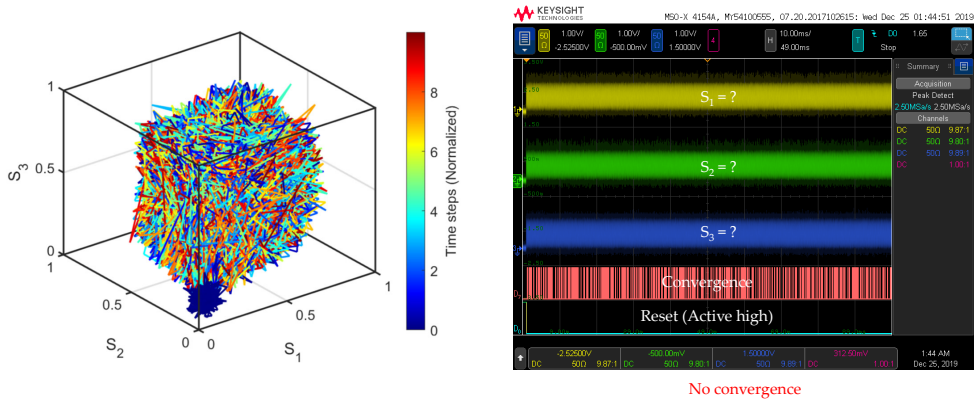


Figure 4.13: Hard and unsolvable case.

As previously mentioned, for the easy one (Figure 4.10), we can see on from the oscilloscope as well that the system almost converges as soon as the reset is pulled down. For the medium one (Figure 4.11), the system searches for a little bit but still converges fairly fast. For the hard but solvable one (Figure 4.12) however, the system searches for pretty long but eventually converges. For the hard and unsolvable one (Figure 4.13), the system becomes chaotic and never finds the solution.

4.3.3 3-SAT Difficulty versus Complexity of Transient Trajectory

To show how the difficulty of the problems relates to the chaotic level, we conducted the detailed measurements on 200 3-SAT problems and collected the transient data as well as the convergence time. We then sorted the result according to the normalized convergence time and analyze the complexity of transient trajectory we were probing, in this case S_1, S_2, and S_3, as shown in the Figure 4.14 (Left). Depending on the interests there may be different ways of quantizing the complexity of transient trajectory, here we use standard deviation of the transient data to quantize the complexity. We can see that the problem complexity is highly positive related to the complexity of transient trajectory.

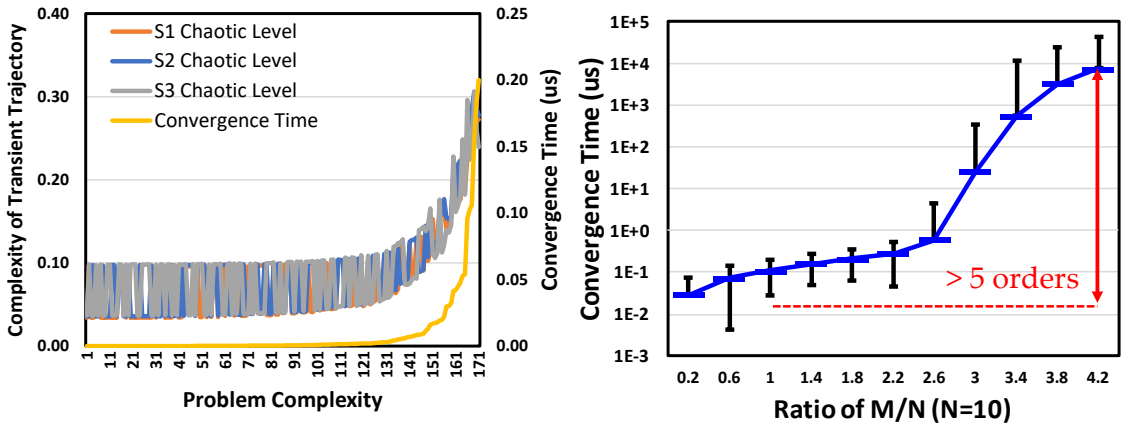


Figure 4.14: (Left) Problem Complexity Versus Complexity of Transient Trajectory. (Right) Convergence time versus different M/N ratio.

4.3.4 Convergence Time versus Different M/N Ratio

To investigate the effectiveness of how AC-SAT performs on the silicon as well as the power consumption of the circuit, we conducted a series of measurements using the standard benchmark k-SAT and the result was collected using 1000 problems for each different problem set containing different number of variables. As shown in Figure 4.14 (Right), as M being the number of clauses and N being the number of variables, we can clearly see that as the ratio of M/N increases from 0.2 to 4.2, the convergence time dramatically increases as well, which is expected as problems with constraint density $M/N = 4.25$ are considered to be hard problems.

4.3.5 Solvability versus Number of Variables

In the Figure 4.15 (a), with all problems being the most difficult problems (M/N=4.25), we tested 1000 problems each on N=10, 20, and 30. As mentioned in the section B, once a_m reaches VDD, the system starts randomly searching inside the possible solution space and therefore is the major physical limitation of such architecture. While we notice a dramatic solvability degradation after N=30, there may be couple reasons causing this. One is as just mentioned that once the auxiliary variable reaches the VDD, the system enters the phase

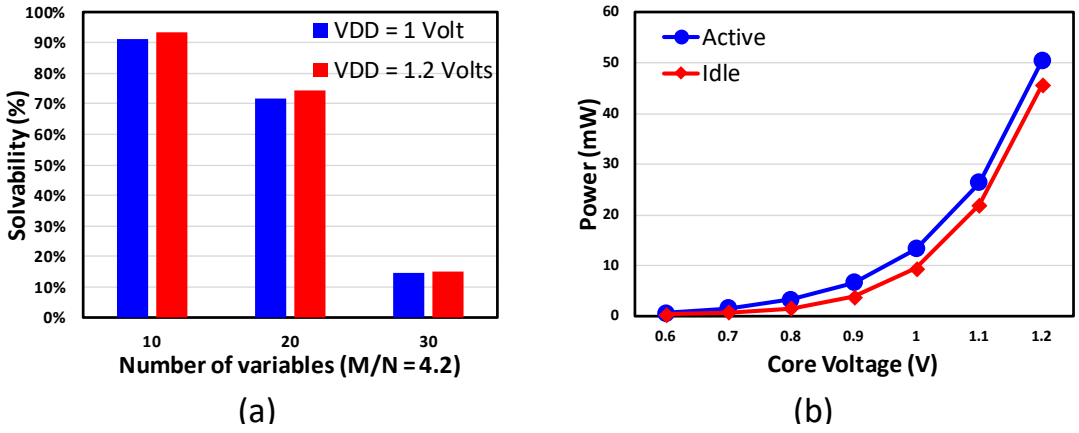


Figure 4.15: (a) Solvability versus number of variables. (b) Power consumption under different core voltages.

where it starts randomly searching in the solution space and since it is not strictly following the mathematical equations, it is not guaranteed to find the solution. Interestingly though, here we show that by increasing the core voltage from 1 volt to 1.2 volts, since the physical limitation of the auxiliary variable is slightly relieved, we do see some very hard problems being solved under 1.2 volts but not 1 volt, with the solvability being slightly improved. Another reason which is a more engineering reason is that as mentioned in the section A, though the blocks are highly modular and scalable, in order to maximize the solvability, the tuning on the sizes of the transistors and the capacitors are required. Depending on the limitation of the chip area as well as the computing resource, it may be extremely difficult and time consuming to find the optimal values of the transistors and the capacitors.

4.3.6 Power Consumption under Different Core Voltages

In Figure 4.15 (b), we measured the power consumption under different core voltages when the system is in either active mode or idle mode, where active mode indicates that the system is making progress and searching for the solutions, and the idle mode means the solution has been found. We can tell two things from the figure, one is that the power increases quadratically as the core voltage increases, and the second one is that the extra power required for the system to be in the active mode is fairly minor comparing to the power consumption in the idle mode. If we again relate the power consumption with the solvability shown in the Figure 4.15 (a), one may naturally prefer 1 volt over 1.2 volts for the core voltage, as trading the power consumption over solvability seems pretty worth it.

4.3.7 Design Challenges

In this section, we want to talk about two main design challenges we encountered during the design phase. One is improving scalability through paying extra attention pitch matching between components, and the other one is the sizing of transistor and the load capacitors.

During the circuit design phase, one particular design challenge we faced was floor

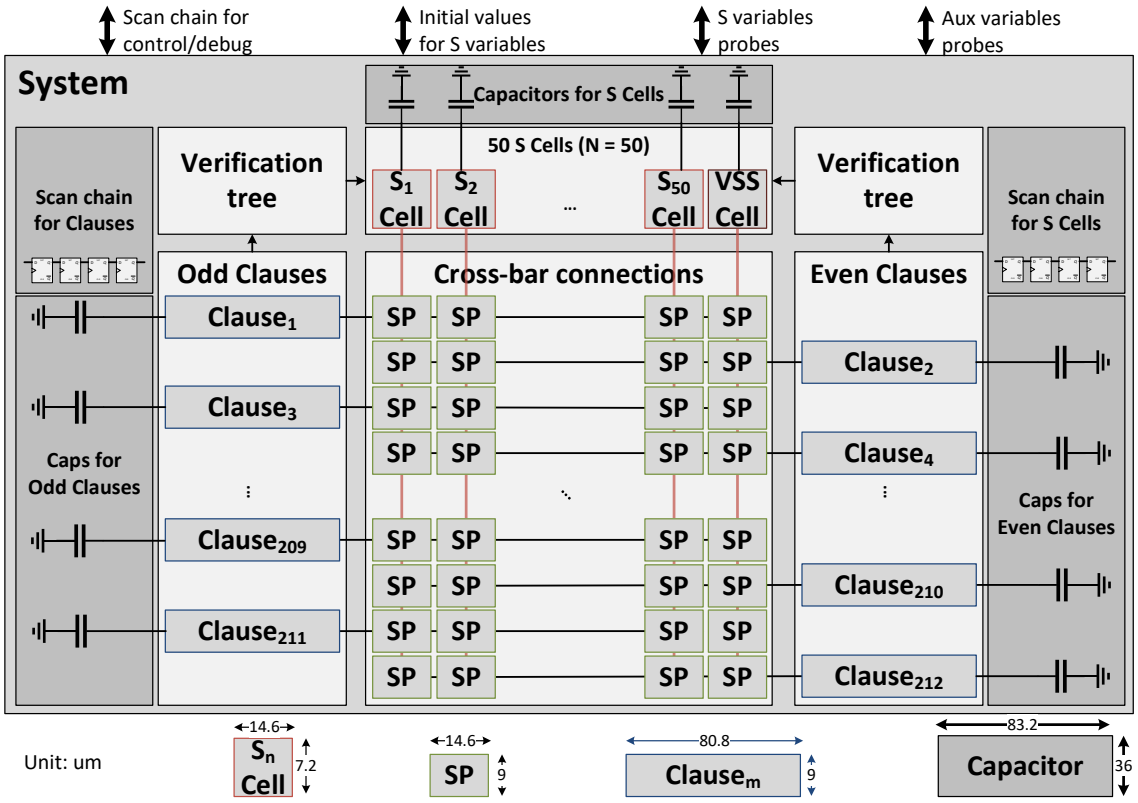


Figure 4.16: Floorplan and the pitch matching.

planning. Since on-die capacitors occupies a lot of area and the shape of them is not as flexible as other component circuits, once we start placing the components, how we design and implement them becomes critical in terms of how many variables and clauses we can fit, and may even cause the extra difficulty to do the routing. In Figure 4.16, we show a rough floor plan how the chip was designed along with the dimension of each component shown at the bottom of the figure. In order to make it straightforward to place them, we match the pitch, as can be seen in the figure, the width of the S cells and the switch pairs matches, and so does the height of the clause and the switch pair. As just mentioned, since the on-die capacitor is not as flexible and in order to ease the routing overhead, we ended up staggering the clauses with odd numbers on the left side and even numbers on the right side of the chip. Because of this carefully designed architecture, it becomes trivial and simple to scale up to more variables and clauses.

Another challenge during the circuit design phase is sizing. In many scenarios the parasitic resistance and capacitance of the wire loads can be neglected, however in this kind of cross-bar connection architecture, the parasitic capacitance due to the long wires can heavily impact the performance. Therefore two things are critical, the first one is the driving strength of each SDC, meaning each SDC needs to have enough strength to drive the long wires, and the second being the value of the load capacitors since we want the equivalent capacitance each clause see to be more or less similar. Imagine if the load capacitors are not large enough, for the scenario where some variables only show once or twice in the problem and some appear many more times will cause the equivalent capacitance to be very unbalanced between those clauses. This technique is common in a lot of analog circuit design area to lower the impact of parasitic capacitance.

CHAPTER 5

DISCUSSION

5.1 Applications for Continuous Optimization Solvers

For continuous optimization problems, we proposed programmable and iterative optimization solvers which are capable of addressing multiple applications. We have implemented

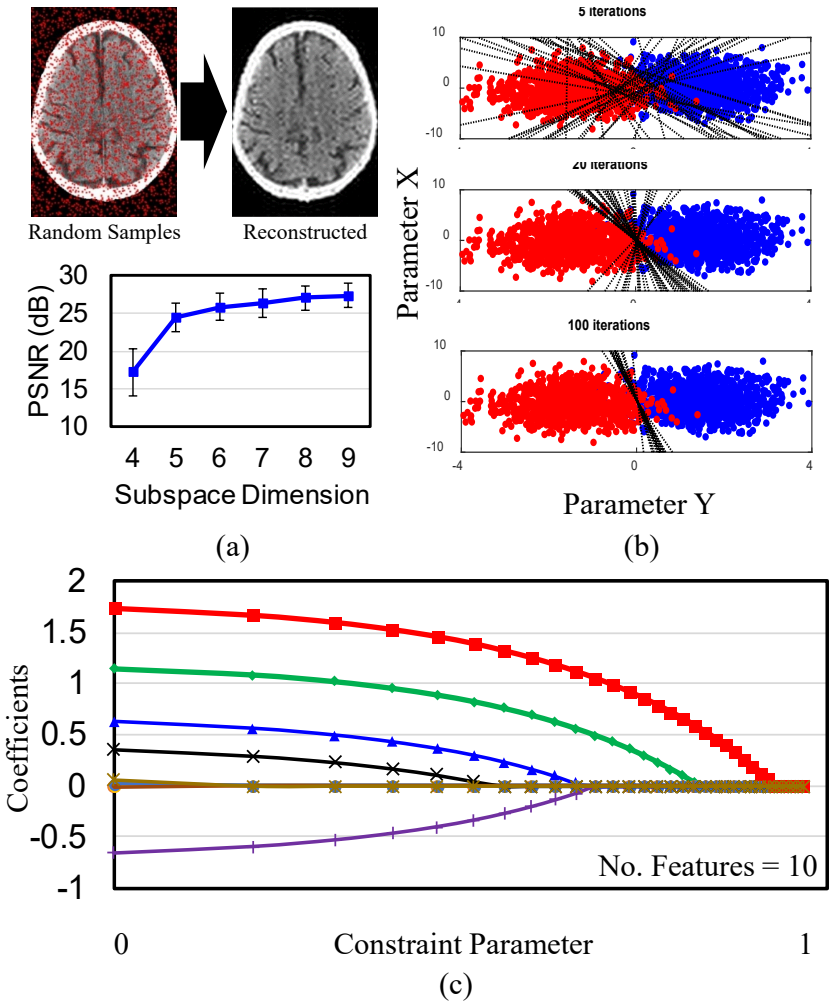


Figure 5.1: Application of OPTIMO in (a) MRI image reconstruction (b) Binary SVM (c) Lasso feature extraction for sample problems.

	This work	[6]	[3]	[4]	[5]	[2]
Application	Distributed Optimization	ECG Signal Reconstruction	CNN Inference	DNN Inference	CNN Inference	CNN Inference
Optimization algorithm	ADMM implementation	subspace pursuit	none	none	none	none
Technology	65nm	40nm	180nm	65nm	65nm	65nm
Area	12mm^2	3.06mm^2	3.3mm^2	16mm^2	16mm^2	16mm^2
On-die SRAM	306.25 KB	192KB	144 KB	36 KB	490.5 KB	181.5 KB
Programming support	yes	fixed function	fixed function	fixed function	fixed function	fixed function
On chip network	8 neighbors with hierarchical multicast	not reported	systolic (4 neighbor)	not reported	systolic (4 neighbor)	systolic (6 neighbor)
Resolution	16b	32b	4b-16b	16b	16b	16b
Power	3.63 - 143.2 mW	21.8 - 93 mW	7.5-300 mW	45 mW	6.57 mW	278 mW
Frequency	10 - 270 MHz	67.5 MHz	200 MHz	125 MHz	10 - 100 MHz	200 MHz
Supply voltage	0.5-1V	0.9V	1V	1.2V	0.7-1.2V	0.82-1.17V
Performance/Watt	0.279 TOPS/W	21.5 MOPS/W	0.26-10TOPS/W	1.42TOPS/W	11.8 - 19.7 GOPS	0.21TOPS/W

Figure 5.2: Comparison of the proposed array-processor with competitive spatial-array processors. The proposed design addresses distributed optimization which presents a more complex data-flow and compute than traditional CNN and DNN inference architectures.

such solvers on FPGA as well as through ASIC. MRI image reconstruction from non-uniformly sampled data-points is computationally challenging and requires patients to lie in the machine for a long time. Our solution uses iterative least-squares optimization (Figure 5.1 (a)) to reconstruct MRI images with high peak signal-to-noise ratio (PSNR) in less than 8ms. Similarly binary SVM (Figure 5.1 (b)), a popular choice in ML classification problems shows convergence with increasing number of iterations (multi-class SVM records 91% accuracy on the MNIST data-set, which is the state-of-the-art). Further, feature extraction with LASSO (L1 regularization) used in ML, is shown in Figure 5.1 (c). In Figure 5.2, a comparison with the state-of-the-art shows a (1) a highly-programmable, iterative optimization solver with peak efficiency of 279GOPS/W (2) a hierarchical multicast network for program-specific data-movement and (3) competitive energy-efficiency and voltage-scalability.

	2	4	
1			3
4			2
	1	3	

Figure 5.3: A 4x4 example of Sudoku puzzle.

5.2 Applications for Combinatorial Optimization Solvers

For combinatorial optimization, we proposed a continuous-time dynamical system where we use k-SAT as our template problem. We implement such system through a fully customized ASIC. There are many applications which can potentially solved through k-SAT, in fact in Karp's 21 NP-Complete problems, all the problems can be reduced to k-SAT. In modern days, one of the popular and interesting applications is Sudoku. Sudoku puzzle is a kind of Latin square and the aim is to complete a $n * n$ grid with legal digits in each square so that, in a Latin square, each digit occurs exactly once in each row and in each column, as shown in Figure 5.3. Though solving Sudoku by SAT is not very appealing for human players, it works well on a computer. Due to the hardware constraints, particularly on the number of clauses required after transferring the Sudoku problem into SAT format, we were able to solve the 4x4 Sudoku problems depending on the difficulty of the problem and whether the number of clauses exceeds the hardware constraints of the testing chip.

5.3 Future Research Opportunities

One of the most discussed topic throughout my research has been the scalability, and the research on the scalability has been widely conducted from multiple perspectives. Each of them comes with its own charm and design considerations.

On the device side, Back-End-of-Line compatible transistors has been of great interest for many researchers because of it's potential to open a completely new design methodology. Thorugh the dynamic random access memory manufacturers have adopted stacked capacitors that tower above the silicon plane and the nand flash memory technologists can alreaedy stack 128 layers of charge trap flash cells on top of each other in a monolithic fashion, to enable monolithic three-dimensional integration of high-performance logic, solving the fundamental challenge of low temperature in situ synthesis of high mobility n-type and p-type semiconductor thin films which can be utilized for fabrication of back-end-of-line

(BEOL) compatible complementary MOS transistors under the constraint of limited thermal budget has been a challenging task. In [46], the authors discuss recent progress in the selection and optimization of semiconductor materials for BEOL compatible transistors to enable sequential M3D integration for a range of applications.

On the packing side, chiplets has been very popular as well as the methods to connect them. In [47], through multi-height and fine-pitch compressible microinterconnect (CMI), the authors demonstrate a polylithic integration technology called heterogeneous interconnect stitching technology (HIST) to integrate multiple chiplets together in 2.5-D and 3-D. In [48], the authors demonstrate the use of metal electroless plating along with mechanical self-alignment as a method to create high density scalable interconnects between chiplets using a low temperature process flow.

On the manufacturing side, wafer scale integration is becoming more and more feasible. One of the most famous company recently would be Cerebras where in 2019 IEEE Hot Chips 31 Symposium they presented $46225mm^2$ silicon containing 1.2 trillion transistors for deep learning [49].

CHAPTER 6
CONCLUSION

Iterative dynamical systems form computational models for solving distributed optimization problems. Due to the semi-separable nature, they can often be parallelized, albeit with architectural support.

In the first part of the thesis, We focus on continuous optimization and propose such a distributed architecture for solving leat-square minimization, which employs a globally asynchronous design by exploiting a 4-phase handshaking mechanism between cores to eliminate unnecessary latency, bandwidth, and energy overheads associated with a global clock. The design is verified on a Xilinx FPGA and measurements reveal that the proposed architecture is scalable to large data-sets.

We then move on the ASIC design and present a 49-core fully-programmable spatial array processor for solving distributed optimizations with support for a large class of algorithms and applications through ADMM. We present a full-stack solution which enables full-programmability, a key requirement for future high-performance systems that need to solve a large class of similar problems. We note a peak performance of 270MHz and peak energy-efficiency of 279 GOPS/W.

In the last part of the thesis, we switch to the other side of optimization, combinatorial optimization, and present a proof-of-principle analog system implemented on silicon using CMOS technology, AC-SAT, based on CTDS to solve 3-SAT problems, and more importantly through this system we demonstrated the relationship between optimization hardness and transient behavior. AC-SAT is modular, programmable and can be used as a SAT solver coprocessor.

Appendices

APPENDIX A

EXPERIMENTAL EQUIPMENT

The following equipments are used throughout the measurement:

- Multimeter
  - Agilent 34461A
  - PXI-4072
- Oscilloscope
  - Agilent MSO9404A
  - Agilent MSO-X 2022A
  - Agilent MSO-X 4154A
  - Agilent MSO-X 4154A
- Power Supply
  - Agilent E3646A
  - Agilent N6700B
  - Keysight E36102A
- Signal Generator
  - Agilent 81134A
  - Agilent 81150A
  - PXIe-6556
- Field Programmable Gate Arrays (FPGAs)

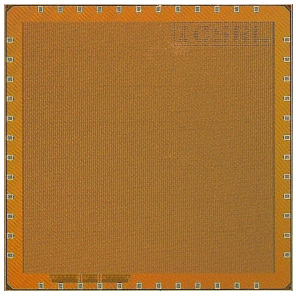
  - Xilinx Virtex-7 FPGA VC707
  - Xilinx Virtex UltraScale+ HBM VCU128
- Arduino Boards
  - UNO
  - MEGA 2560
  - DUE

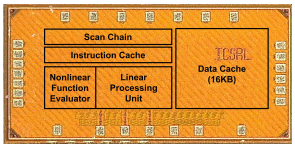
APPENDIX B

CHIP GALLERY

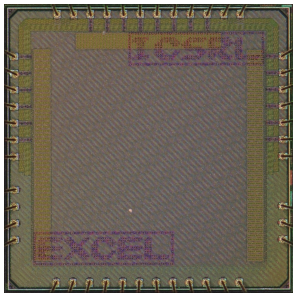
Designing circuits and taping out have been a very crucial part in the past four years. From knowing nothing to being able to complete the whole tape-out flow and also being able to help others complete the same is really an accomplishment to myself. Not only did I learn how to design the circuit, but also learned how to solve the problems and conquer the difficulties.

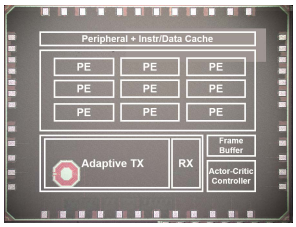
Throughout the program I have been honored to have so many opportunities to be part of so many projects and participate in the process. In this section I list all the integrated circuits (IC) that I had been a part of. The pleasure was all mine to have such chances to work with all the talented colleagues: Ningyuan, Jonghyeok, Xunzhao, Brian, and Sam.



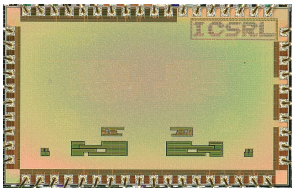


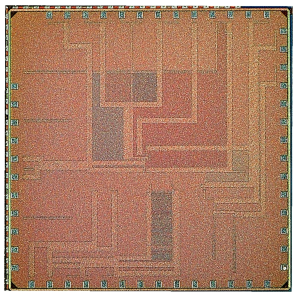
Domain	Optimization	**Domain**	Edge Computing
Name	OPTIMO	**Name**	Swarm
Technology	65nm GP 1P9M	**Technology**	65nm GP 1P9M
Tape-out Date	6th June 2018	**Tape-out Date**	6th June 2018
Die Size	3.41 x 3.14 mm^2	**Die Size**	2.00 x 1.00 mm^2
Package	QFN6x6-48	**Package**	QFN4x4-28
Publication	CICC; JSSC	**Publication**	ISSCC; JSSC



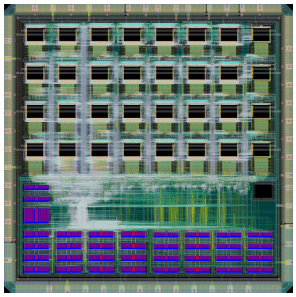


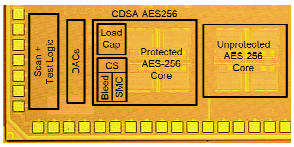
Domain	Optimization	**Domain**	Edge Computing
Name	AC-SAT	**Name**	COOPS
Technology	65nm GP 1P9M	**Technology**	65nm GP 1P9M
Tape-out Date	10th April 2019	**Tape-out Date**	4th May 2019
Die Size	2.54 x 2.54 mm^2	**Die Size**	2.00 x 2.50 mm^2
Package	COB	**Package**	COB
Publication		**Publication**	VLSI





Domain	Power Converter	**Domain**	Emerging Technology
Name	GAN DC DC	**Name**	RRAM
Technology	180nm HV BCD	**Technology**	40nm ULP 1P6M
Tape-out Date	15th May 2019	**Tape-out Date**	14th Aug 2020
Die Size	4.00 x 2.50 mm^2	**Die Size**	3.00 x 3.00 mm^2
Package	COB	**Package**	COB
Publication		**Publication**	ISSCC





Domain	Emerging Technology	**Domain**	Encryption
Name	PCM	**Name**	CDSA-AES256
Technology	40nm LP 1P6M	**Technology**	65nm LP 1P9M
Tape-out Date	10th Oct 2020	**Tape-out Date**	Jun 2019
Die Size	3.00 x 3.00 mm^2	**Die Size**	1.30 x 0.50 mm^2
Package	COB	**Package**	COB
Publication		**Publication**	ISSCC; JSSC

REFERENCES

[1] Z. Gui-Xia, Z. Cheng-Jing, and W. Xiao-Yan, "Research of distributed data optimization storage and statistical method in the environment of big data," in *2017 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, 2017, pp. 612–617.

[2] A. L'Heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz, "Machine learning with big data: Challenges and approaches," *IEEE Access*, vol. 5, pp. 7776–7797, 2017.

[3] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: A new platform for distributed machine learning on big data," *IEEE Transactions on Big Data*, vol. 1, no. 2, pp. 49–67, 2015.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[5] A. Parihar, N. Shukla, M. Jerry, S. Datta, and A. Raychowdhury, "Vertex coloring of graphs via phase dynamics of coupled oscillatory networks," *Scientific Reports*, vol. 7, 2017.

[6] A. Ibrahim, T. F. Al-Somani, and F. Gebali, "New systolic array architecture for finite field inversion," *Canadian Journal of Electrical and Computer Engineering*, vol. 40, no. 1, pp. 23–30, 2017.

[7] A. Ibrahim, F. Gebali, and T. F. Al-Somani, "Systolic array architectures for sunar–koç optimal normal basis type ii multiplier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 10, pp. 2090–2102, 2015.

[8] R. Wimalagunarathne, A. Madanayake, D. G. Dansereau, and L. T. Bruton, "A systolic-array architecture for first-order 4-d iir frequency-planar digital filters," in *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2012, pp. 3069–3072.

[9] J. Zhang, W. Zhang, G. Luo, X. Wei, Y. Liang, and J. Cong, "Frequency improvement of systolic array-based cnns on fpgas," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1–4.

[10] M. Krstic, E. Grass, F. K. Gürkaynak, and P. Vivet, "Globally asynchronous, locally synchronous circuits: Overview and outlook," *IEEE Design Test of Computers*, vol. 24, no. 5, pp. 430–441, 2007.

[11] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[12] Y. Chen, T. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[13] M. Huang and H. Wang, "21.2 a 27-to-41ghz mimo receiver with n-input-n-output using scalable cascadable autonomous array-based high-order spatial filters for instinctual full-fov multi-blocker/signal management," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 346–348.

[14] A. Biswas and A. P. Chandrakasan, "Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 488–490.

[15] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42pj/decision 3.12tops/w robust in-memory machine learning classifier with on-chip training," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 490–492.

[16] X. Si, J. Chen, Y. Tu, W. Huang, J. Wang, Y. Chiu, W. Wei, S. Wu, X. Sun, R. Liu, S. Yu, R. Liu, C. Hsieh, K. Tang, Q. Li, and M. Chang, "24.5 a twin-8t sram computation-in-memory macro for multiple-bit cnn-based machine learning," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 396–398.

[17] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, and D. Sylvester, "14.2 a compute sram with bit-serial integer/floating-point operations for programmable in-memory vector acceleration," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 224–226.

[18] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno, "24.3 20k-spin ising chip for combinational optimization problem with cmos annealing," in *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, 2015, pp. 1–3.

[19] T. Takemoto, M. Hayashi, C. Yoshimura, and M. Yamaoka, "2.6 a 2 ×30k-spin multichip scalable annealing processor based on a processing-in-memory approach for solving large-scale combinatorial optimization problems," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 52–54.

[20] X. Yin, B. Sedighi, M. Varga, M. Ercsey-Ravasz, Z. Toroczkai, and X. S. Hu, "Efficient analog circuits for boolean satisfiability," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 1, pp. 155–167, 2018.

[21] H. Mostafa, L. K. Müller, and G. Indiveri, *An event-based architecture for solving constraint satisfaction problems*, 2015.

[22] H.-T. Kung, "Why systolic architectures?" *IEEE computer*, vol. 15, no. 1, pp. 37–46, 1982.

[23] S. Borkar, H. Kung, *et al.*, "Supporting systolic and memory communication in iwarp," in *Computer Architecture, 1990. Proceedings., 17th Annual International Symposium on*, IEEE, 1990, pp. 70–81.

[24] J. G. Nash, "High-throughput programmable systolic array fft architecture and fpga implementations," in *Computing, Networking and Communications (ICNC), 2014 International Conference on*, IEEE, 2014, pp. 878–884.

[25] S. D. Muñoz and J. Hormigo, "High-throughput fpga implementation of qr decomposition," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 9, pp. 861–865, 2015.

[26] J. McWhirter, "Recursive least-squares minimization using a systolic array," *Real-Time Processing VI, Aug. 1983*, pp. 105–112, 1983.

[27] P. J. S. Ferreira, "The stability of a procedure for the recovery of lost samples in band-limited signals," *Signal Processing*, vol. 40, no. 2-3, pp. 195–205, 1994.

[28] R. Marks, "Restoring lost samples from an oversampled band-limited signal," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 31, no. 3, pp. 752–755, 1983.

[29] H. Stark, "Polar, spiral, and generalized sampling and interpolation," in *Advanced Topics in Shannon Sampling and Interpolation Theory*, Springer, 1993, pp. 185–218.

[30] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. now, 2011, ISBN: 9781601984609.

[31] N. R. Draper and H. Smith, *Applied regression analysis*. John Wiley & Sons, 1998, vol. 326.

[32] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[33] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 1, pp. 49–67, 2006.

[34] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the royal statistical society: series B (statistical methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[35] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[36] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[37] M. Ercsey-Ravasz and Z. Toroczkai, *Optimization hardness as transient chaos in an analog approach to constraint satisfaction*, Oct. 2011.

[38] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, May 1983.

[39] M. R. Garey, D. S. Johnson, and R. Sethi, *The complexity of flowshop and jobshop scheduling*, Jan. 1976.

[40] Y. Xie and A. Aiken, "Scalable error detection using boolean satisfiability," *SIGPLAN Not.*, vol. 40, no. 1, pp. 351–363, Jan. 2005.

[41] S. Zhang and A. G. Constantinides, "Lagrange programming neural networks," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 7, pp. 441–452, 1992.

[42] M. Nagamatu and T. Yanaru, *On the stability of lagrange programming neural networks for satisfiability problems of prepositional calculus*, Jun. 1999.

[43] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[44] M. Chang, L.-H. Lin, J. Romberg, and A. Raychowdhury, "Optimo: A 65nm 270mhz 143.2mw programmable spatial-array-processor with a hierarchical multi-cast on-chip network for solving distributed optimizations," *IEEE Custom Integrated Circuits Conference*, 2019.

[45] J. Ax, N. Kucza, M. Vohrmann, T. Jungeblut, M. Porrmann, and U. Rückert, "Comparing synchronous, mesochronous and asynchronous nocs for gals based mpsocs,"

in *2017 IEEE 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, 2017, pp. 45–51.

[46] S. Datta, S. Dutta, B. Grisafe, J. Smith, S. Srinivasa, and H. Ye, "Back-end-of-line compatible transistors for monolithic 3-d integration," *IEEE Micro*, vol. 39, no. 6, pp. 8–15, 2019.

[47] P. K. Jo, S. Kochupurackal Rajan, J. L. Gonzalez, and M. S. Bakir, "Polylithic integration of 2.5-d and 3-d chiplets enabled by multi-height and fine-pitch cmis," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 10, no. 9, pp. 1474–1481, 2020.

[48] S. K. Rajan, M. J. Li, M. S. Bakir, and G. S. May, "High density and low-temperature interconnection enabled by mechanical self-alignment and electroless plating," in *2019 International 3D Systems Integration Conference (3DIC)*, 2019, pp. 1–4.

[49] "Wafer-scale deep learning," in *2019 IEEE Hot Chips 31 Symposium (HCS)*, 2019, pp. 1–31.

VITA

EDUCATION

Ph.D in Electrical & Computer Engineering (Dec 2020) at Georgia Institute of Technology. Thesis title: "Hardware Dynamical System for Solving Optimization Problems."

M.S in Computer Science (May 2020) at Georgia Institute of Technology.

B.S in Electronic Engineering (May 2014) at National Chiao Tung University, Taiwan.

ACADEMIC EMPLOYMENT

Graduate Teaching Assistant, College of Engineering, Georgia Institute of Technology, Aug 2016 - Dec 2018. Responsibilities include: assisting professors with the preparation and presentation of undergraduate and graduate courses, grading, and tutoring.

Research Assistant to Prof. Arijit Raychowdhury, College of Engineering, Georgia Institute of Technology, Aug 2016 - Dec 2020.

PUBLICATIONS

(Under review) J. Yoon, M. Chang, A. Raychowdhury, "A 40nm 64Kb 56.67TOPS/W Ternary Compute-in-Memory RRAM Macro with Voltage-sensing Read and Write Verification for reliable multi-bit RRAM encoding", IEEE CICC 2021.

(Under review) J. Yoon, M. Chang, W. Khwa, Y. Chih, M. Chang, A. Raychowdhury, "Reliable Write Operations and Multi-bit Encoding in high-endurance RRAM arrays", IEEE IRPS, 2021.

J. Yoon, M. Chang, W. Khwa, Y. Chih, M. Chang, A. Raychowdhury, "A 40nm 64Kb 56.67TOPS/W Read-Disturb-Tolerant Compute-in-Memory/Digital RRAM Macro with Active-Feedback-Based Read and In-Situ Write Verification", IEEE ISSCC, 2021.

M. Gong, N. Cao, M. Chang, A. Raychowdhury, "A 65nm Thermometer-Encoded Time-Based Compute-in-Memory Neural Network Accelerator at 0.735pJ/MAC and 0.41pJ/Update," IEEE TCAS-II, 2020.

D. Das, M. Chang, A. Raychowdhury, S. Sen et al, "EM and Power SCA-resilient AES-256 through >350×Current Domain Signature Attenuation & Local Lower Metal Routing," IEEE JSSC, 2020.

N. Cao, B. Chatterjee, M. Gong, M. Chang, S. Sen, A. Raychowdhury, "A 65nm Image Processing SoC Supporting Multiple DNN Models and Real-Time Computation-Communication Trade-Off Via Actor-Critical Neuro-Controller," IEEE Symposium on VLSI Circuits, 2020.

D. Das, M. Chang, A. Raychowdhury, S. Sen et al, "EM and Power SCA-Resilient AES-256 in 65nm CMOS Through >350x Current-Domain Signature Attenuation," IEEE ISSCC, 2020.

M. Chang, L. Lin, J. Romberg, and A. Raychowdhury, "Optimo: A 65nm 279gops/w 16b programmable spatial-array-processor with on-chip network for solving distributed optimizations via the alternating direction method of multipliers," IEEE JSSC, 2019.

N. Cao, M. Chang, and A. Raychowdhury, "A 65nm 8-3b 1.0-0.36v 9.1-1.1tops/w hybriddigital-mixed-signal computing platform for accelerating swarm robotics," IEEE JSSC, 2019.

I. Yoon, M. Chang, A. Raychowdhury et al. "A ferrofet based in-memory processor for solving distributed and iterative optimizations via least-squares method," IEEE JXCDC, 2019.

M. Chang, S. Gangopadhyay, T. Hamam, J. Romberg, and A. Raychowdhury, "Efficient signal reconstruction via distributed least square optimization on a systolic fpga architecture," in 2019 IEEE ICASSP

M. Chang, L. Lin, J. Romberg, and A. Raychowdhury, "65nm 49core processor array with hierarchical multicast on chip network for solving distributed optimizations," in 2019 IEEE CICC

N. Cao, M. Chang, and A. Raychowdhury, "14.1 a 65nm 1.1-to-9.1tops/w hybrid-digital-mixedsignal computing platform for accelerating model-based and model-free swarm robotics," in 2019 IEEE ISSCC

I. Yoon, M. Chang, A. Raychowdhury et al. "A fefet based processing-in-memory architecture for solving distributed least-square optimizations," in 2018 76th DRC

ACADEMIC AWARDS

Taiwan Government Scholarship to Study Abroad (GSSA), May 2019

Qualcomm Innovation Fellowship Award, May 2019

Chih Foundation Graduate Student Research Publication Award, May 2019

ECE Graduate TA Excellence Award (GaTech), Apr 2019